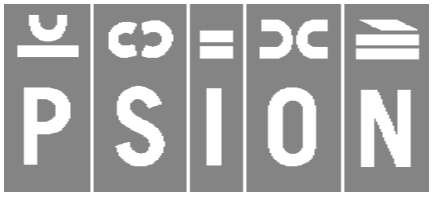
OPL

FROM OPL16 INTO OPL32: A GUIDE TO PORTING TO OPL32



CONTENTS

OPL

INTRODUCTION

The principal design requirements of OPL32 were:

- Compatibility with OPL16, also known as SIBO OPL.
- Provision of a mechanism for language extensions to replace direct calls to the operating system.
- Generally to take advantage of the abilities of EPOC32.

This document describes the changes to OPL16. It is a porting guide and not a tutorial. See one of the OPL16 manuals, such as the Series 3c Programming manual, for information about aspects of the OPL language that have not changed in OPL32.

REMOVED OPL16 FEATURES

The following OPL16 features have been omitted or replaced in OPL32:

- TYPE, PATH and EXT. These were provided on SIBO for use by the System screen only. The Series 5 System screen does not require this information. On the Series 5, applications do not have types, application-specific paths or filename extensions. An application and its associated documents are identified by a UID (unique identifier) instead as discussed below. (TYPE has been replaced by FLAGS - see below.)
- STATUSWIN, STATWININFO, DIAMINIT, DIAMPOS. The Series 5 has no status windows.
- OS, CALL. OPL32 provides extensibility using special OPL DLLs (see OPXs below).
- USR, USR$.
- ODBINFO. This is implementation specific (see above).
- LOADLIB, LINKLIB, UNLOADLIB, FINDLIB, GETLIBH, NEWOBJ, NEWOBJH, SEND, ENTERSEND, ENTERSEND0. A different mechanism is now used for calling language extensions and creating object instances.
- CACHE, CACHETIDY, CACHEHDR, CACHEREC. OPL32 procedures are automatically cached.
- Named Calculator memories and M0,...,M9. The Series 5 Calculator does not use OPL to evaluate expressions.
- CREATESPRITE, APPENDSPRITE, CHANGESPRITE, DRAWSPRITE, POSSPRITE, USESPRITE, CLOSESPRITE. Superior sprite-handling OPX functions are provided.
- RECSIZE and COMPRESS. COMPACT keyword replaces COMPRESS.
- gINFO i%() is replaced by gINFO32 i&().
- CMD$(4) and CMD$(5).
- SETNAME. The new SETDOC, serving a similar purpose, should be called before saving your main document.
- gDRAWOBJECT.

THE PROGRAM EDITOR

The name of the OPL program editor application is Program. See the User Guide for details.

To convert an OPL16 text file to OPL32, you will need to:

- Create a new Program document from the System screen, either using the toolbar button or the File|Create new|File... menu command.
- Import the text using File|More|Import text...

Conventionally you would not use a filename extension for the Program source file.

SYSTEM-LEVEL CHANGES

PRIORITY KEYS: PAUSE AND KILL KEYS

On SIBO, the 'Kill' key was Psion+Esc, Control+S paused and any other key resumed.

OPL32 has different keypresses for WINS and the Series 5, to avoid clashes with the Windows task list keys. For OPL32, the 'Kill' key is:

WINS: Shift+Esc

Series 5: Control+Esc

and the 'Pause' and 'Resume' keys are:

WINS: Control+Alt+S and Control+Alt+Q

Series 5: Control+Fn+S and Control+Fn+Q

respectively.

MODIFICATION OF CHARACTER CODES

On SIBO, pressing the Psion key (or the Alt key on the PC), added $200 to the value of the keycode. This is no longer true for the Series 5. However, as on Sibo, the keycode value is modified when pressing Ctrl at the same time as another key. For alphabetic keys the Ctrl modified value is the ordinal position in the alphabet ignoring upper and lower case, i.e. 1 for Ctrl+A or Shift+Ctrl+A, 2 for Ctrl+B or Shift+Ctrl+B, etc. This value can be found by ANDing the Ascii value of the alphabetic character with (NOT $60). So the keycode of upper and lower case letters is the same when pressed together with Ctrl, with only the value of the modifier differing (see below). For example, pressing Ctrl+J returns keycode 10 which is (%j AND (NOT $60)).

KEYBOARD MODIFIERS

Keyboard modifier values are unchanged from OPL16, except that the Psion key (modifier value 8) is no longer used. GETEVENT, KEYA etc., GETEVENT32 saves the modifiers in ev&(4) for keypresses and in ev&(5) for pointer (pen) events. See "General new keywords" for details.

The new Fn modifier has code 32.

From Const.oph,

```
CONST KKmodShift%=2
CONST KKmodControl%=4
CONST KKmodCaps%=16
CONST KKmodFn%=32
```

UIDS AND DOCUMENTS

On the Series 5, the word “document” has a special meaning. A document is a file that launches its associated application when it is selected. The application UID is stored in the document header which is read by the system. A non-document file does not have an application UID and is displayed on the system screen with a special icon (a question mark icon) showing that it is unrecognised. Non-document files are known as *external files*.

For OPL applications, the argument uid& in the APP declaration specifies the UID of the application. For applications that are to be distributed, UIDs are reserved by contacting Psion. These official UIDs are guaranteed to be unique to the application and have values of &10000000 and above. For applications developed for personal use there is no need to reserve official UIDs, and any UID between &01000000 and &0fffffff may be selected. If UIDs of two applicatuions clash, then either your application’s documents will have the wrong icon and selecting these will run the other application, or the other application’s documents will seem to belong to your application. See the sections describing APP and SETDOC for further details of OPL applications.

OPL32 explicitly supports database and multi-bitmap files as documents. To create a document rather than an external file, use SETDOC to set the current document name to the file that is to be created, e.g. on receiving the ‘Create file’ command on the command-line (cmd$(3)=“C”), you would use:

```
SETDOC cmd$(2)
CREATE cmd$(2),a,...
```

SYSTEM SCREEN COMPLIANCE

A well-behaved Series 5 application should follow the style guide which is also available as part of this SDK.

Some important points:

- All applications should respond in the standard way to the initial command-line and to any System screen messages to switch files. See GETEVENT32 and GETCMD$ and changes to CMD$(3) for details.
- All applications should ensure that they handle corrupt .INI files correctly. When starting up with CMD$(3)=”R”, try and open the .INI file. If this fails for **any** reason, just fall back to using defaults - DO NOT report the error to the user (who will not understand the error or know how to fix it).
- All Series 5 applications should support a toolbar. Toolbar.opo is supplied in the ROM and provides the set of procedures required for this support. Toolbar.opo is documented fully below.
- All Series 5 applications should save non-document files (external files) to their so-called *application directory*. For an application called AppXxx, the application directory is \System\Apps\AppXxx\. This is also the directory where the application itself is saved.

  You can find out the full file specification of external files by using PARSE$. For example,

  ```
  p$=PARSE$(“NEW”,LEFT$(CMD$(1),LEN(CMD$(1)-4),x%())
  ```

  CMD$(1) gives the device and the path of the OPL application and the name of the file.
- By default the system screen hides the System directory and its subdirectories. Use Tools|Preferences menu command (Control+K) in the System screen and check the checkboxes for both ‘Show hidden files’ and ‘Show “System” folder’ to show all applications and their files.

- The FLAGS keyword in the APP...ENDA construct should have the value KFlagsAppFileBased% (1) if the application supports documents, where the word “document” is used in the sense defined above (a file that will run your application when selected from the System screen). Without this flag the application will not be listed when the user chooses to create a new file from the System screen.

- An application should ser ESCAPE OFF so that the user cannot break out of the application at any time by using Ctrl+Esc.

LAUNCHING HELP FILES

It is possible to write your own Help for your own application, which should be created in the Data application. The file should be similar to the built-in help, using two labels only for each entry to give a title and the explanation itself. The file should be stored in the application’s folder with the filename extension .hlp. The procedure RUNAPP&: in System OPX may then be used to launch Data and open the application help file. For example, if you want to run a help file called Myapp.hlp, the help file for your own application called Myapp, you should use:

```
drv$=LEFT$(CMD$(1),2)
RUNAPPP&:(“Data”,drv$+“\System\Apps\Myapp\Myapp.hlp”,“”,0)
```

You should make it possible for users of your application to launched your help file from the application’s menus. Following the Series 5 style guide, the ‘Help’ item should appear on the ‘Tools’ menu at the bottom of the section above ‘Infrared’, so it will be followed by a separating line. The shortcut key should be Ctrl+Shift+H.

GENERAL FEATURES OF OPL32

This section outlines features of OPL32 that are not keyword-specific and that are not related to graphics and database handling.

Some major conceptual, though largely OPL16-compatible, changes have had to be made in the OPL32 database and graphics model, and these warrant discussion in sections of their own.

HEADER FILES, CONSTANTS AND PROCEDURE PROTOTYPES

OPL32 allows you to include header files which may include definitions of constants and procedure prototypes.

The standard header file Const.oph in the ROM provides many of the standard declarations required for effective and maintainable OPL32 programming. For convenient reference, the contents Const.oph is reproduced in full in an Appendix to this document.

To use Const.oph, add the following at the top of your module:

```
INCLUDE “Const.oph”
```

Const.oph is stored in the ROM folder z:\System\Opl\.

See INCLUDE, CONST and EXTERN for more information.

LONG NAMES

Identifiers (names of procedures, variables, parameters and constants) may now have up to 32 characters including any type-specification character. In addition, the underscore character is allowed in all identifiers, except in the last position when the identifier includes a type-specification character. The first character of any identifier must be either an underscore or an alphabetic character.

POINTER EVENTS

The word ‘pointer’ refers to the pen on the Series 5 or to the mouse in the WINS emulation.

OPL32 supports the handling of pointer (pen or mouse) events and they can be read in a similar manner to other events. See GETEVENT32 and the asynchronous version GETEVENTA32.

FONTS

On the Series 5, fonts are identified by a 32-bit UID, rather than a 16-bit value representing the font position in the ROM as in SIBO. OPL32 does however provide a mapping where possible between OPL16 font IDs and the Series 5 UIDs, but there are inevitably some incompatibilities. Note that, Series 3 Classic fonts 1, 2 and 3 are not supported on the Series 5: these were only supported on the Series 3a and later SIBO machines to enable Series 3 compatible programming.

As well as being able to use the OPL16 font IDs 4 to 13 (which are automatically converted to long integers for all keywords taking font IDs), you can also directly specify the fonts by UID on the Series 5 by using the definitions listed in the header file Const.oph.

For example, normal Arial proportional font with height 8 pixels has UID given by,

```
CONST KFontArialNormal8&=268435954
```

so you could use,

```
INCLUDE “Const.oph”
...
gFONT KFontArialNormal8&
```

Bold Times New Roman proportional font with height 11 pixels has UID

```
CONST KFontTimesBold11&=268435963
```

The Squashed font is used for Toolbar buttons, with bold style.

User-defined fonts may be loaded as in OPL16 using gLOADFONT, but the return value is the font *file* ID. For user-defined fonts, gFONT can evidently only take a font UID. The value returned by gLOADFONT is used by gUNLOADFONT only.

It should be noted that when using the console keywords PRINT, AT, SCREEN, etc. the use of proportional rather than monospaced fonts may produce some unexpected behaviour because it is assumed in all cases that mono font is being used. The reason for this is so that use of keywords such as AT, SCREEN in the console is independent of the font. An example of this behaviour may be seen when using inverted text: the rectangle to invert is calculated assuming that the font is monospaced, and hence the area inverted is larger than the text printed when using a proportional font. Another example is that a new line will be used before it appears necessary when using a proportional font, since the number of characters which will fit on a line is also calculated assuming the font is monospaced.

The default font for PRINT statements is Courier size 11 and the default graphics font (for gPRINT, etc.) is Arial size 15.

MASKS

In OPL32 there are several keywords that require an understanding of so-called masks. In some cases the mask is a bitmap file, e.g. for ICON and gBUTTON, and in some cases it is an integer containing a bit-mask, e.g. for POINTERFILTER.

In all these cases, however, the principle is the same. The bits which are set in the mask specify bits in some other argument which are to be used. Bits which are clear in the mask specify bits that are not to be used from the other argument.

For example, for `ICON mbm$`, bitmaps have to be paired, with the icon's bitmap followed by its mask in the multi-bitmap file mbm$. Only pixels which are set in the mask are drawn in the final icon. The pixels which are clear in the mask are not drawn, allowing the background to be displayed in these pixels. See ICON for further details of its usage.

For `POINTERFILTER filter%,mask%` bits that are clear in mask% are left alone, whether they are zeroes or ones. This makes it possible to set some flags, clear some others and leave some alone all in one call to POINTERFILTER. To set or clear flags you would set and clear them in filter% as required and set all the flags that require changing in mask%. To leave some bits in filter% alone just don't set the bits in mask%. See POINTERFILTER for further details of its usage.

OPX LANGUAGE EXTENSIONS

OPL32 uses language extensions provided in separate EPOC32 DLLs written especially for OPL support. These DLLs can be added to the language by anyone at any time and have the file extension '.OPX'. Unlike procedures written in OPL, these procedures are as fast to call as built-in keywords. OPXs enable OPL programs to perform virtually any operation in EPOC32 which is available to a C++ program. OPX procedures are called in a similar way to user-defined OPL procedures.

32-BIT ADDRESSING

OPL32 has no built-in memory limits and an application can use as much of the available memory as it requires. In OPL16, heap allocations, variables and OPL object code were restricted to using 64K of memory. This placed a strong constraint on the processing power of OPL16 applications.

Memory addresses outside the 64K address space need to be stored in long integers. Thus, the syntax of all keywords that support addresses has changed accordingly:

```
addr&=ADDR(variable)
addr&=ALLOC(size&)
newCell&=REALLOC(oldCell&,size&)
newCell&=ADJUSTALLOC(oldCell&,off&,amount&)
FREEALLOC cell&
err%=IOOPEN(var h%,addr&,mode%)      (for the case of unique file creation only)
err%=IOREAD(h%,addr&,maxlen%)
err%=IOWRITE(h%,addr&,length%)
PEEK and POKE keywords               (e.g. i%=PEEKW(addr&), POKEW addr&,i%)
```

Compare these with the equivalent OPL16 keywords.

Note also that where '#' was used to specify the address of a variable that is listed in the syntax using 'var', the address is now a long integer, so you would use #address& rather than #addr%.

For example, the syntax of IOSEEK is unchanged from OPL16, namely,

```
ret%=IOSEEK(h%,mode%,var offset&)
```

However, when you use '#', you would not call IOSEEK using,

```
ret%=IOSEEK(h%,mode%,#ptrOff%)
```

as in OPL16, but instead,

```
ret%=IOSEEK(h%,mode%,#ptrOff&)
```

passing the long integer ptrOff&.

In OPL32, these functions should use 32-bit integers for storing the address value. Otherwise 'Overflow' errors will occur when trying to store an address beyond the 64K limit in a 16-bit integer. So, for example, where SIBO code might have used,

```
p%=ALLOC(bytes%)
```

in OPL32 you would normally use,

```
p&=ALLOC(bytes&)
```

To facilitate porting to OPL32, it is in fact possible to set a flag to emulate the OPL16 memory model using `SETFLAGS KRestrictTo64K&`. This will cause an 'Out of memory' error if an attempt is made to obtain an address beyond the 64K limit. If this flag is set, OPL32 checks that the 64K limit is not exceeded when,

- the variables for a procedure are allocated, on calling any procedure. If the address of any variable in the procedure would require more than 16 bits, the procedure will fail to be loaded and an 'Out of memory' error will be raised. This is evidently preferable to having, for example, p%=ADDR(i%) giving an 'Overflow' error, once the procedure has been loaded. The ideal solution is, of course, to port the program to use 32-bit addresses instead, but setting the flag is a quick solution for many applications that are known to require less than 64K.

- the value returned by the heap allocation keywords requires more than 16 bits.

Some existing OPL16 programs may at times attempt to exceed the 64K limit and deal with the 'Out of memory' error. To port such programs there are two choices. Either you can change all integers that contain addresses to be 32-bit or set the flag that enforces the 64K limit using SETFLAGS.

Note that the use of the word "address" in OPL32 is in fact imprecise. OPL32 "addresses" are in fact offsets into OPL's variable-heap, so an "address" of 0 really means 0 offset relative to the base of OPL's heap. This allows OPL32 to perform the appropriate 64K limit tests. This method of addressing variables is referred to as "base-relative addressing".

(Another potential problem may well concern you at this point. When, for example, displaying an OPL32 address in some diagnostic debugging code, if a variable i% or a heap cell has a base-relative offset of between 32768 and 65535, then `p%=ADDR(i%)` would produce an overflow error unless special measures were taken. The reason for this is that 16-bit integers are signed in OPL, so the range is -32768 to 32767. Hence, although 32768 fits in an unsigned 16-bit integer (hex $8000) it doesn't fit into a signed integer. OPL32 gets around this problem by *sign-extending* the result of ADDR and the heap functions (only when 64K restriction is in force) as follows:

- if the value returned is 32768 to 65535 (hex $8000 to $ffff), OPL32 treats the value as signed. So $8000 becomes &ffff8000 (or -32768), and $ffff becomes &ffffffff (or -1). These values can then be assigned to the signed p%.

- if the value is greater than or equal to 65536 (hex &10000) then it is not sign-extended and so assigning it to p% raises an 'Overflow' error as required.

As mentioned above, this conversion should always be totally transparent to your program. This is possible because the keywords that use these sign-extended addresses, all convert the value back to an unsigned 16-bit value before use.)

There is one case, however, where porting is required even with the flag set for a 64K limit. This is when the long integer value returned by ADDR or from a heap-allocating function is passed directly into a user-defined procedure. An OPL16 procedure taking an address parameter will probably have been written to take a 16-bit address. As the translator cannot necessarily know the type taken by a user-defined procedure, it cannot coerce the type returned by these functions to match that taken by the user-defined procedure. This means that a type violation error can be caused.

Example:

If a user-defined procedure, userProc:(p%) is called as follows:

```
userProc:(ADDR(i%))
```

the 32-bit integer returned by ADDR will cause a type violation, as it will not be coerced to a 16-bit integer. The solution is either to define userProc: to take a 32-bit integer, or to declare the prototype of userProc: as userProc:(ADDR%). Either of these would allow translator coercion. See EXTERNAL for more information on procedure prototyping in OPL32.

ALLOC AND ASSOCIATED HEAP KEYWORDS

ALLOC, REALLOC and ADJUSTALLOC allocate cells that have lengths that are the smallest multiple of four greater than the size requested. All of these now raise errors if the cell address argument is not in the range known by the heap. The same address checking is done for peeking and poking, but note that OPL32 allows the addresses of the application-specific SIBO magic statics DatApp1 to DatApp7 (hex 28 to 34 inclusive) to be used for compatibility.

These keywords now return a 32-bit value, so that a request can be made to allocate a cell of any length within memory constraints. In OPL16 the limit was less than 64K.

MAXIMUM DATA SIZE IN A PROCEDURE

Although there is no built-in limit to the total amount of data in an OPL32 program, it is still not possible to declare variables in a procedure that use in total more than 65516 bytes. This allows the largest integer array in a procedure to have 32757 elements. This number decreases for any other variables, externals or for procedures called from the given procedure, as they all consume some of the procedure’s data space.

The maximum in OPL16 was about 32758 bytes of data in total per procedure.

ERRORS ADDING ITEMS TO DIALOGS

In OPL32, if an error occurs when adding an item to a dialog, the dialog is deleted and dINIT needs calling again. This is necessary to avoid having partially specified dialog lines.

In practical terms, this means that where the following artificial example would have worked in OPL16, giving just a long integer editor,

```
dINIT
  ONERR e1
  dCHOICE ch%,"ChList","a,b,,,,c" rem bad arg list gives argument error
e1::
  ONERR OFF
  dLONG l&,"Long",0,12345
DIALOG
```

it will raise a 'Structure fault' error in OPL32.

NEW ERROR CODES AND ERROR MESSAGES (ERR□AND ERR$, ERRX$)

EPOC32 error codes are mapped on to E16 error codes where possible. New Series 5-specific errors that cannot be mapped to existing errors have been assigned new codes in the range -121 upwards, defined in Const.oph as follows,

```
CONST KErrOpxNotFound%=-121          REM OPX not found
CONST KErrOpxVersion%=-122           REM Incompatible OPX version
CONST KErrOpxProcNotFound%=-123      REM OPX procedure not found
CONST KErrStopInCallback%=-124       REM STOP used in call-back from OPX
CONST KErrIncompUpdateMode%=-125     REM Incompatible update mode
                                     REM (see Database keywords)
CONST KErrInTransaction%=-126        REM In database transaction or
                                     REM started changing fields
```

The new function

```
x$=ERRX$
```

returns the current extended error message "Error in *module\procedure,extern1,extern2,...*" which would have been presented as an alert if the error had not been trapped. This allows the list of missing externals, missing procedure names, etc. to be found when an error has been trapped by a handler.

EPOC16 provided a larger set of error codes than that of EPOC32. This is because it is considered an Application's responsibility to provide its own contextual error messages. So in some cases you may find that the OPL16 error codes returned by certain keywords do not exactly match the OPL16 codes.

I/O KEYWORDS

IOOPEN, IOCLOSE, IOC, IOW, IOWAIT, IOWAITSTAT and IOYIELD

The I/O keywords provide the same access to files as in OPL16, but only the SIBO device drivers "TIM:", "TTY:A", "CON:" and "FIL:" are emulated. You can use handles -1 for the LOPENed device and -2 for "CON:" as in OPL16.

IOYIELD must always be called before polling status words, i.e. before reading a 16-bit status word if IOWAIT or IOWAITSTAT have not been used first. This is because OPL32 needs to set the 16-bit status word based on the value of the actual EPOC32 32-bit status word. In SIBO this was unnecessary for some device drivers and servers.

CONSOLE SERVICES

Many of the console services which were supported in OPL16 are no longer supported in OPL32. In general this is because there are keywords which perform similar services. The services which are still supported are as follows,

- FSET (=7), SCR_SCROLL (=1) Scroll the window content
- FSET (=7), SCR_CLR (=2) Clear a rectangle
- FSET (=7), SCR_SLOCK (=6) Set scroll lock
- FSET (=7), SCR_WLOCK (=7) Set auto wrap
- FSET (=7), SCR_NEL (=8) Position cursor to next line
- FSET (=7), SCR_LAST_LINE_WRAP
- FSENSE (=8) Sense console data
- FINQ (=12) Get console data

These functions work as in OPL16 with the exception of FINQ. As FINQ cannot return the 32-bit font ID (see above)as a 16-bit integer, it no longer returns the font id at all, the second element of the passed array being left unused. See also SCREENINFO.

The following services are no longer supported:

- FSET, SCR_WSET
- FSET, SCR_POSA
- FSET, SCR_POSR
- FSET, SCR_CURSOR
- FSET, SCR_ESCAPE
- FSET, SCR_CSET
- FSET, SCR_ATTRB
- FSET, SCR_FONT
- FSET, SCR_FLUSH
- FSET, SCR_DISABLE_READS
- FSET SCR_CLIENT_FOREGROUND
- FSET, SCR_CAPTURE_KEY
- FSET, SCR_CANCEL_CAPTURE_KEY
- FFLUSH
- EVENT_READ
- EVENT_TEST
- FWFLUSH

FILENAME EXTENSIONS

There are no longer any default file name extensions for Application documents. (System filename extensions, such as APP, OPO, AIF, INI and OPX, still exist). OPL no longer adds default file extensions when using certain keywords as it did in OPL16, such as:

.ODB	for OPEN, CREATE etc.
.PIC	for gLOADBIT, gSAVEBIT.
.FON	for gLOADFONT.

Any extensions may be used but files (documents) in the system are now recognised by UIDs (see above).

In fact, the filing system does not treat extensions as a special component of a filename except when parsing (i.e. for PARSE$) where the extension is treated as normal. In EPOC16 “myfile.” means the same as “myfile” (without the dot), however, in EPOC32 trailing dots are stripped and hence trying to refer to a file named “myfile.” will not find any file, even if a file was initially given this name. You can also use long file names with embedded spaces and any number of dots like: “OPL Reference manual” or “Very.long.filename”.

RANGE OF FLOATING-POINT VARIABLES

In OPL32 it is now possible to use the full avaliable range of 64-bit floating-point values, i.e. all real numbers with absolute values in the range 2.2250738585072015E-308 to 1.7976931348623157E+308 and 0. Precision remains limited to about 15 significant figures in this range. It is also possible to use numbers which have absolute values in the range 5E-324 to 2.2250738585072015E-308 (called “denormals”), however the precision decreases in this range to only 1 significant figure at the lower end.

CURRENT DIRECTORY

The current directory for all commands is always c:\ unless it has been changed by the command SETPATH. Hence any use of a keyword which takes a filename as an argument will only look in the current directory and so if this is other than c:\, it should be specified either by SETPATH or by including it in the filename. For example, to check whether the file “Program1” in the directory “d:\MyPrograms\” exists, either

```
SETPATH “d:\MyPrograms\”
...
IF EXISTS (“Program1”)
...
```

or

```
IF EXISTS (“d:\MyPrograms\Program1”)
...
```

For OPL16, the current directory was “M:\OPD\”

TOOLBAR USAGE

The toolbar on the Series 5 replaces the Sibo status window. All OPL32 programs should use the ROM module z:\System\Opl\Toolbar.opo to create a toolbar window with a title, four buttons and a clock.

The public interface to TOOLBAR.OPO is supplied in z:\System\Opl\Toolbar.oph which is reproduced below. The procedures and their usage are then discussed in detail.

TOOLBAR.OPH

```
rem Toolbar.oph version 1.1
rem Header file for OPL's toolbar
rem (c)Copyright Psion PLC 1997

rem Public procedures
external TBarLink:(appLink$)
external TBarInit:(title$,scrW%,scrH%)
external TBarSetTitle:(name$)
external TBarButt:(shortcut$,pos%,text$,state%,bit&,mask&,flags%)
external TBarOffer%:(winId&,ptrType&,ptrX&,ptrY&)
external TBarLatch:(comp%)
external TBarShow:
external TBarHide:

rem The following are global toolbar variables usable by Opl programs
rem or libraries. Usable after toolbar initialisation:
rem TbWidth%    the pixel width of the toolbar
rem TbVis%  -1 if visible and otherwise 0
rem TbMenuSym%   current 'Show toolbar' menu symbol (ORed with shortcut)

rem Flags for toolbar buttons
const KTbFlgCmdOnPtrDown%=$01
const KTbFlgLatchStart%=$12    rem start of latchable set
const KTbFlgLatchMiddle%=$22   rem middle of latchable set
const KTbFlgLatchEnd%=$32      rem end of latchable set
const KTbFlgLatched%=$04       rem set for current latched item in set
                               rem End of Toolbar.oph
```

TYPICAL TOOLBAR.OPO USAGE

Typically a program would use TOOLBAR.OPO in the following way:

- LOADM "z:\System\Opl\Toolbar.opo" to load the toolbar module. This module must remain loaded as long as you need to use the toolbar.
- 'Link' the toolbar-specific globals into the top-level of your program, using TBarLink:
- Initialise the toolbar, creating an invisible toolbar window with title and clock, using TBarInit:
- Add normal or latchable toolbar buttons to the toolbar, using TBarButt:
- Make the toolbar visible, using TBarShow:
- Offer all pointer events to the toolbar so that it can call your commands, change the system clock type or

display the task list, using TBarOffer%:

- Latch a button down to represent the current view when the view changes, using TBarLatch:
- Change the toolbar title to the current document name, using TBarSetTitle:
- Show and hide the toolbar as appropriate, using TBarShow: and TBarHide:
- Provide *command-handling* procedures to be called by TOOLBAR.OPO when a toolbar button is pressed.

TBARLINK:

Usage: TBarLink:(appLink$)

TBarLink: provides all toolbar globals required in your application. It has to be called before TBarInit: and from a higher level procedure in your application than the globals are used.

appLink$ is the name of the so-called *continuation procedure* in your main application. TBarLink: calls this procedure, which should then go on to run the rest of your program. This allows the globals declared in TBarLink: to exist until the application exits. appLink$ must represent a procedure with name and parameters like:

```
proc appContTBarLink:
   rem Continue after 'linking' toolbar globals
   myAppInit:   rem run rest of program
   ...
endp
```

i.e. taking no parameters and with no return-type specification character, so that it can be called using `@(appLink$):`.

TBARINIT:

Usage: TBarInit:(title$,scrW%,scrH%)

Called at start of application only, this procedure creates the toolbar window, which guarantees that there will be sufficient memory available to display the toolbar at any subsequent time. The toolbar is made invisible when not shown. This procedure also draws all toolbar components except the buttons.

Note that, for speed, TBarInit: turns graphics auto-updating off (using gUPDATE OFF). If automatic updating is required, use gUPDATE ON after TBarInit: returns.

Note also that TBarInit: sets compute mode off (see SETCOMPUTEMODE: in System.oxh) allowing the program to run at foreground priority when in foreground. By default OPL programs have compute mode on (i.e. they run at background priority even when in foreground).

title$ is the title shown in the toolbar. You should change this to the name of your current file, using TBarSetTitle:.

scrW% is the full-screen width (gWidth at startup).

scrH% is the full-screen height (gHeight at startup).

TBARSETTITLE:

Usage: TBarSetTitle:(name$)

Sets the title in the toolbar.

name$ is the name of your current file for file-based applications (i.e. applications with the APP...ENDA construct containing FLAGS 1), or the name of your application for non-file applications.

TBARBUTT:

Usage: TBarButt:(shortcut$,pos%,text$,state%,bit&,mask&,flags%)

Adds a button to the previously initialised toolbar.

shortcut$ is the command shortcut for your application, which is used by TOOLBAR.OPO to perform the command when a toolbar button is selected. On selecting the toolbar button, TOOLBAR.OPO calls your procedure to perform the required command or action. shortcut$ is case sensitive in the sense that TOOLBAR.OPO calls your procedure named:

- "cmd"+shortcut$+%: for unshifted, lower-case shortcuts,
- "cmdS"+shortcut$+%: for shifted, upper-case shortcuts.

For example, if you have the following two commands that also have associated toolbar buttons:

```
mCARD "View", "DoXXX",%x, "DoYYY",%Y    rem shortcuts Ctrl+X,
      Shift+Ctrl+Y
```

you would need to provide command-handling procedures:

- `PROC cmdX%: rem handle shortcut Ctrl+X (lower case shortcut$="x")`
- `PROC cmdSY%:  rem handle shortcut Shift+Ctrl+Y (upper case shortcut$="Y")`

and you would create buttons using, e.g.:

- `TBarButt:("x",1, "DoXXX",0,XIcon&,XMask&,0)  rem Button for calling cmdX:`
- `TBarButt:("Y",1, "DoYYY",0,YIcon&,YMask&,0)  rem Button for calling cmdSY:`

pos% is the button position, with pos%=1 for the top button.

text$ and state% take values as required for gButton.

bit& and mask& are the button's icon bitmap and mask, used in in the same way as for gButton.

flags% lets you control how the button is used in two distinct and mutually exclusive ways, as follows:

1. Two *latchable* buttons are often used by the built-in applications to indicate the current view. For an example, see the latchable 'Desk' and 'Sci' buttons in the Calculator. The style guide describes in detail how latchable buttons are intended to be used.
   A set of latchable toolbar buttons can be specified in TBarButt: by setting flags% to one of:
   KTbFlgLatchStart% for the first button in the latchable set.
   KTbFlgLatchMiddle% for any middle buttons (these are optional and not generally used).
   KTbFlgLatchEnd% for the last button in the latchable set.

To latch a button down initially to represent the initial setting, OR KTbFlgLatched% with one of the above settings. E.g.

TBarButt:(sh1$,pos%,txt1$,st%,bit1&,msk1&,KTbFlgLatchStart%)
TBarButt:(sh2$,pos%,txt2$,st%,bit2&,msk2&,KTbFlgLatchEnd% OR KTbFlgLatched%)

will latch down the second button in the set initially.

In the toolbar window, the button with KTbFlgLatchStart% set must be above the buttons (if any) with KTbFlgLatchMiddle% set, and these in turn must be above the button with KTbFlgLatchEnd%.
Only one button in a set is ever latched and pressing another button unlatches the one that was previously set. After pressing and releasing a previously unlatched button in a latchable set, TOOLBAR.OPO will, as usual, call your command-handling procedure. When the command has succeeded in changing view, this procedure should set the new state of the button by calling TBarLatch:(comp%) where comp% is the button number to be latched. This will also unlatch any button that was previously latched. The example below shows how a 'View1' button press, with "v" as shortcut, should be handled. The other latchable button in this set might be 'View2' with shortcut "w":

```
proc cmdV%:
  if SetView1%:=0   rem if no error
    TBarLatch:(KView1TbarButton%) rem your const KView1TBarButton%
    CurrentView%=1
  endif
endp
proc cmdW%:
  if SetView2%:=0   rem if no error
    TBarLatch:(KView2TbarButton%) rem your const KView2TBarButton%
    CurrentView%=2
  endif
endp
```

You should call the same command-procedures when the command is performed via a menu or via a keyboard shortcut. This will ensure that the button is latched as required.

2. A setting of flags% can also be used to specify that your procedure should be called when the toolbar button is touched (rather than when the button is released, which is the default). The 'Goto' button in the Program editor works in this way, displaying the popup list of procedures when the button is touched. To implement this using TBarButt:, pass flags%=KTbFlgCmdOnPtrDown% and provide a procedure named: "cmdTbDown"+shortcut$+%: which could provide a popup menu, as follows:
proc cmdTbDownC%:

```
rem popup next to button, with point specifying top right corner of
    popup
  if mPopup(ScrWid%-
    TbWidth%,97,KMPopupPosTopRight%,"Cancel",0,"Clear",%c)
  cmdC%: rem Do the command itself
  endif
endp
```

In this case the shortcut is not case-sensitive. Note that when this flag is used, the menu command-procedure is not used directly because a popup is not required when the command is invoked via the menu or via a keyboard shortcut.

TBAROFFER%:

Usage: TBarOffer%:(winId&,ptrType&,ptrX&,ptrY&)

Offers a pointer event to the toolbar, returning -1 if used and 0 if not used. If not used, the event is available for use by your application.

It is important to call this procedure whenever you receive a pointer event, even when the event is not in the toolbar window, thus enabling TOOLBAR.OPO to release the current button, both visually and otherwise.

TBarOffer%: handles:

- The tapping of the clock to change the type between analog and digital system-wide.
- The tapping of the toolbar title to display the task list.
- The selection toolbar buttons, calling your command procedures.
- Drawing of the button in the appropriate state.

As usual, the word ‘pointer’ indicates a pen on the Series 5 or the mouse in the WINS emulation.

winId& is the ID of the window that received the pointer event.

ptrType& is pointer event type as returned by GETEVENT32 (up,down or drag).

ptrX&,ptrY& is coordinate of pointer event.

TBARLATCH:

Usage: TBarLatch:(button%)

Latches down a toolbar button, where button%=1 for the top button in the toolbar. TBarLatch: also unlatches any button in the latchable set that was previously latched. See TBarButt: for further details on latching buttons.

TBARSHOW:

Usage: TBarShow:

Makes the toolbar visible. The toolbar must exist before calling this procedure. Use TBarInit: to create an invisible toolbar with no buttons. Use TBarButt: to add buttons.

TBARHIDE:

Usage: TBarHide:

Makes the toolbar invisible.

PUBLIC TOOLBAR GLOBALS

The following toolbar globals, provided by TBarLink:, may be used in OPL32 applications:

- TbWidth% is the pixel width of the toolbar. The rest of the screen width is available for the application.
- TbVis% is -1 if visible and otherwise 0
- TbMenuSym% is the current ‘Show toolbar’ menu symbol to be ORed with menu shortcut for View|Show toolbar, e.g.:

  ```
  mCard “View”,”Show toolbar”,%t or TbMenuSym%
  ```

  TbMenuSym%=(KMenuCheckBox% OR KMenuSymbolOn%) if the Toolbar is visible and TbMenuSym%=KMenuCheckBox% if invisible. The menu item will therefore be a checkbox item, with the check present or not as appropriate.

GENERAL CHANGED KEYWORDS

This section provides an alphabetic listing of all non-graphics and non-database OPL16 keywords which have been changed in some way for OPL32. (See separate sections for graphics and database changes.)

ADDR

Usage: `addr&=ADDR(variable)`

The address returned is now a long integer.

See SETFLAGS if you require the 64K limit to be enforced. If the flag is set to restrict the limit, addr& is guaranteed to fit into a short integer.

ALLOC

Usage: `addr&=ALLOC(bytes&)`

The number of bytes allocated is no longer restricted to 64K as in OPL16. The return type is therefore a long.

See SETFLAGS if you require the 64K limit to be enforced. If the flag is set to restrict the limit, addr& is guaranteed to fit into a short integer.

APP

Usage: `APP` *`caption`*`,uid&`

caption is the application's name (or caption) in the machine's default language. Note that although it is a string it is not enclosed in quotes. This name is displayed in the System screen's task list. It is also used by the Extras bar itself and as the default document name for documents lauched using the Extras bar. Any use of the CAPTION keyword (see below) causes this default caption to be discarded, as well as specifying different captions for different languages.

uid& is the application UID as described in detail in the section 'System-level changes' above. This 32-bit value uniquely identifies your application and its documents from all others on the system and provides the System Screen with the information it requires to run your application when the user selects a document or the application itself.

The APP...ENDA construct is used by the translator to generate an Application Information File (AIF) containing 3 icon and mask pairs, the application caption and the setting of FLAGS. The AIF file is created in the application directory together with the APP file itself. E.g. The AIF for AppXxx.app would be:

\System\Apps\AppXxx\AppXxx.aif

The translator will generate an AIF based on all the information available in the APP…ENDA construct. However, if any is missing, then defaults will be used. There are:

- for ICON: the default (question mark) icons
- for CAPTION: the caption specified in the APP declaration
- for FLAGS: the default value of 0.

BUSY

Usage: Unchanged from OPL16

The maximum string length of a BUSY message is now 80 characters and an “Invalid argument” error is returned for any value in excess of this.

CMD$(3)

Usage: `c$=CMD$(3)`

In addition to CMD$(3) returning “C” or “O” it may now also return “R”.

“R” means that your application has been run from the Program editor or has been selected via the Application’s icon on the Series 5 ‘Extras’ bar, and not by the selection or creation of one of your documents from the System screen. A default filename, including path, is passed in CMD$(2).

When “R” is passed, an application should always try to open the file last used when it last exited. This filename should be stored in a .INI file with the same name and in the same directory as your application. So for example, AppXxx.app would have INI file \System\Apps\AppXxx\AppXxx.ini.

The last-used file is the file that was in use when the application was closed, not the file that was most recently opened. This means that the file last viewed is the one that is loaded when the application is next selected.

If the INI file does not exist, or if the file listed there as the last file no longer exists, you should create the file named in CMD$(2). CMD$(2) is a default name provided by the System screen based on your application’s caption. See CAPTION for further details. If the INI file is corrupt, it should be detected before going on to create CMD$(2). No error message should be displayed in this case.

CURSOR

Usage: Unchanged from OPL16

The CURSOR keyword is in general the same as in OPL16, except that the obloid cursor is no longer supported, i.e. type%=1 no longer gives an obloid cursor, but just displays a default graphics cursor, as though no type had been specified.

DBUTTONS

Usage: Unchanged from OPL16

Const.oph supplies these two constants,

```
CONST KDButtonNoLabel%=$100
CONST KDButtonPlainKey%=$200
```

To display a button with no shortcut key label underneath it, OR the flag KDButtonNoLabel% with the shortcut. To use the key alone (without the Ctrl modification) as the shortcut key, OR the flag KDButtonPlainKey% with the shortcut. When using a negative shortcut to specify the cancel button (as in OPL16), you must negate the shortcut together with any flags as shown below.

E.g. These buttons for a Cancel/Confirm dialogs will not have “Esc” and “Enter” displayed beneath them:

```
dBUTTONS “Cancel”,-(27 OR KDButtonNoLabel%),”Confirm”,%c OR KDButtonNoLabel%
```

and these buttons for a Yes/No dialog will use Y and N (rather than Ctrl+Y and Ctrl+N) as their shortcut keys:

```
dBUTTONS “Yes”,%Y OR KDButtonPlainKey% OR KDButtonNoLabel%,”No”,-(%N OR
        KDButtonPlainKey% OR KDButtonNoLabel%)
```

DCHOICE

Usage: `dCHOICE var choice%,p$,list1$+”,...”`

```
dCHOICE var choice%,””,list2$+”,...”
...
dCHOICE var choice%,””,listN$
```

dCHOICE now supports an unrestricted number of items (up to memory limits). To extend a dCHOICE list, add a comma after the last item on the line followed by “...” (three full-stops), as shown in the usage above. choice% must be the same on all the lines, otherwise an error is raised. For example, the following specifies items i1,i2,i3,i4,i5,i6:

```
dCHOICE ch%,prompt$,”i1,i2,...”
dCHOICE ch%,””,”i3,14,...”
dCHOICE ch%,””,”i5,i6"
```

DFILE

Usage: `dFILE f$,prompt$,flags%[,Uid1=0,Uid2=0,Uid3=0]`

Const.oph supplies the following constants for dFILE:

```
REM Opl-related UIDs
CONST KUidOPLInterpreter&    = 268435816
CONST KUidOPLApp&            = 268435572
CONST KUidOPLDoc&            = 268435573
CONST KUidOPO&               = 268435571
CONST KUidOPLFile&           = 268435594
CONST KUidOpxDll&            = 268435549

REM Flags
CONST KDFileEditBox%                  =$0001
CONST KDFileAllowFolders%             =$0002
CONST KDFileFoldersOnly%              =$0004
CONST KDFileEditorDisallowExisting%   =$0008
CONST KDFileEditorQueryExisting%      =$0010
CONST KDFileAllowNullStrings%         =$0020
CONST KDFileAllowWildCards%           =$0080
CONST KDFileSelectorWithRom%          =$0100        REM New flag
CONST KDFileSelectorWithSystem%       =$0200        REM New flag
```

Several changes have been made to dFILE to support the new system architecture:

- dFILE must be passed a string with length 255, since file names may be up to this length.
- By default OPL32 doesn’t display any prompts for the file, folder and disk editors. A comma-separated prompt list should be supplied. For example, for a filename editor with the standard prompts use:

  ```
  dFile f$,”File,Folder,Disk”,1
  ```

- The flags% value $40 to omit file extensions is no longer valid since file extensions are no longer treated as special components of the filename (see “File extensions” above).
- For a filename selector, flags% can have the new flag KDFileSelectorWithRom% ($0100), to allow ROM files to be selected.

- For a filename selector, flags% can have the new flag KDFileSelectorWithSystem% ($0100), to allow files in the System folder to be selected.
- For file selectors, dFILE now supports file restriction by UID (see above), or by type from the user's point of view. Documents are identified by three UIDs which identify by which application the document was created and what kind of file it is. Specifying all three UIDs will sort the files as precisely as is possible, and specifying fewer will provide a broader sort. You can supply 0 for `Uid1&` and `Uid2&` if you only want to restrict the list to `Uid3&`. This may be useful when dealing with documents from one of your own applications: you can easily find out the third UID as it will be the UID you specified in the APP statement. Note that UIDs are ignored for editors.

  UIDs are used to identify files in the system for many purposes. The System screen uses Uid2 to distinguish between applications of various types and documents:

  Eikon applications

  Eikon applications' documents

  OPL applications

  OPL applications' documents

  Uid3 is used by the System screen to identify particular applications.

  For example, if your application has UID KUidMyApp&, then the following will list only your application-specific documents:

```
dFILE f$,prompt$,flags%,0,KUidOplDoc&,KUidMyApp& rem KUidOplDoc& for OPL docs
```

DINIT

Usage: `dINIT`

```
dINIT title$
dINIT title$,flags%
```

flags% can be any ORed combination of the following constants, defined in Const.oph:

```
CONST KDlgButRight%     =1   REM Buttons on right rather than at bottom
CONST KDlgNoTitle%=2         REM No title (any title in dINIT ignored)
CONST KDlgFillScreen%   =4   REM Use the full screen
CONST KDlgNoDrag% =8         REM Don't allow dialog box to be dragged
CONST KDlgDensePack%    =16  REM Pack dialog densely (not buttons)
```

It should be noted that dialogs without titles cannot be dragged regardless of the "No drag" setting. Dense packing enables more lines to fit on the screen for larger dialogs. The remaining flags are self-explanatory.

DTEXT

Usage: `dTEXT prompt$,body$,flags%`

(N.B. flags% is still an optional parameter.)

The following constants are supplied in Const.oph.

```
CONST KDTextLeft%       =0
CONST KDTextRight%      =1
CONST KDTextCentre%     =2
CONST KDTextBold%       =$100  REM Ignored in Eikon
CONST KDTextLineBelow%  =$200
CONST KDTextAllowSelection%=$400
CONST KDTextSeparator%  =$800
```

You can now display a line separator between any dialog items by setting the flag KDTextSeparator% ($800) on a dTEXT item which has null prompt$ and body$. (If prompt$ and/or body$ are not null, then the flag is ignored and no separator is drawn.) The separator counts as an item in the value returned by DIALOG. OPL32 still supports OPL16's underlining of dTEXT items by setting flag KDTextLineBelow% ($200).

It should be noted that OPL32 only supports those features which EIKON supports. This implies that bold dialog text is not supported by OPL32. It also means that alignment of the body$ is only supported when the prompt$ is null, with the body being left aligned otherwise.

Also note that setting the flag KDTextAllowSelection% ($400) only allows the prompt, but not the body, to be selected.

DTIME

Usage: Unchanged from OPL16

The following constants are provided in Const.oph to be used for the t% parameter in dTIME which specifies the type of display:

```
CONST KDTimeAbsNoSecs%           =0
CONST KDTimeWithSeconds%         =1
CONST KDTimeDuration%            =2
CONST KDTimeDurationWithSecs%    =3
CONST KDTimeNoHours%             =4
CONST KDTime24Hour%              =8
```

The values and their meanings are the same as in OPL16, except for the last two which are additions in OPL32. KDTimeNoHours% displays the time without hours and KDTime24Hours% displays the time in 24 hour clock, regardless of the system setting.

Absolute times always display am or pm as appropriate, unless 24 hour clock is specified. Durations never display am or pm. Note, however, that if you use the flag KDTimeNoHours% then the am/pm symbol will be displayed and the flag KDTimeDuration% must be ORed in if you wish to hide it.

DXINPUT

Usage: Unchanged from OPL16

This keyword may not be passed a string longer than 16 characters.

GETEVENT

Usage: `GETEVENT ev%()`

This command responds to all GETEVENT32 events and writes the same codes to ev%(1) as GETEVENT32 ev&() writes to ev&(1), but the array elements ev%(2) to ev%(6) are only set for the OPL16 events handled on Sibo.

It is strongly recommended that you use GETEVENT32 instead. GETEVENT is supported only for backward compatibility and cannot be used to handle pointer events in a satisfactory way.

ICON (AND BMCONV)

Usage: `ICON mbm$`

mbm$ is the name of a multi-bitmap file (MBM), also known as an EPOC Picture file, which contains the icons for an OPL application. The multi-bitmap file can contain up to three bitmap/mask pairs - currently the sizes are 24, 32 and 48 squares. These different sizes are used for the different zoom levels in the system screen. The sizes are read from the MBM and the most suitable size is zoomed if the exact sizes required are not provided or if some are missing.

In fact, you can use ICON more than once within the APP...ENDA construct. The translator only insists that all icons are paired with a mask of the same size in the final ICON list. This allows you to use MBMs containing just one bitmap as produced by the Sketch application.

For example, you could specify them individually:

```
APP ...
  ICON “icon24.mbm”
  ICON “mask24.mbm”
  ICON “icon32.mbm”
  ICON “mask32.mbm”
  ICON “icon48.mbm”
  ICON “mask48.mbm”
ENDA
```

or with pairs in each MBM:

```
APP ...
  ICON “iconMask24”
  ICON “iconMask32”
  ICON “iconMask48”
ENDA
```

or with all the bitmaps as just one MBM, as would normally be the case if prepared on the PC.

MBMs can be produced from a set of BMPs using the PC tool BMCONV.EXE. Type BMCONV at the DOS prompt for the following help screen:

PSION MULTIPLE BITMAP FILE/ROM STORE CONVERTOR PROGRAM.

Usage: BMCONV [/r] [/hheaderfile] psifile [/4]bmpfile_1 [... [/4]bmpfile_n]

Or: BMCONV /u psifile bmpfile_1 [... bmpfile_n]

Or: BMCONV commandfile

/r optionally specifies a ROM image destination file,

the alternative is a File Store destination file.

/hheaderfile optionally specifies the automatic

generation of a header file for inclusion into code.

/4 (no space) creates a 4 bpp file, default is 2 bpp.

psifile specifies the psion multi-bitmap file name.

bmpfile_n specifies the nth win bitmap file name.

/u converts psifile to bmpfile_1,...,bmpfile_n

otherwise bmpfile_1,...,bmpfile_n are compiled to psifile

commandfile specifies a file containing the commandline,

with commands separated by spaces or newlines.

A typical usage would be to call:

BMCONV bld.txt

where bld.txt is the commandfile containing something like:

/hmyapp.mbg myapp.mbm

bmp1.bmp

bmp2.bmp

bmp3.bmp

/4gray16.bmp

The first line provides the BMCONV flags other than the BMPs, with generated header \Epoc32\Include\Myapp.mbg and output file Myapp.mbm. The following lines list the BMPs to be converted, with a leading /4 for 4 bits per pixel, as stated in the help screen.

The header file generated using this commandfile contains C++ enums for inclusion in C++ programs. These provide symbolic names for the MBM index, which can be used for example by gLOADBIT.

```
// myapp.mbg
// Generated by BitmapCompiler
// (C) Copyright Psion PLC 1996
//
enum TMbmMyapp
```

```
{
EMbmMyappBmp1,
EMbmMyappBmp2,
EMbmMyappBmp3,
EMbmMyappGray16
};
```

OPL developers can easily convert these to OPL constants:

```
const KMbmMyappBmp1%=0
const KMbmMyappBmp2%=1
const KMbmMyappBmp3%=2
const KMbmMyappGray16%=3
```

MCARD

Usage: Unchanged from OPL16

All OPL16 menu handling features are still supported in OPL32, though several important features have been added to improve OPL32 menu handling, giving them similar look and feel to menus in non-OPL applications.

To be consistent with other Series 5 manuals and documentation, SIBO hot-keys are now referred to as shortcuts. In OPL32, mCARD, the cascade variant, mCASC and the popup variant mPOPUP (see “General new keywords”) allow several flag values to be ORed with the shortcut keycode for a menu item, as discussed below.

It is very important to ensure that your menus follow certain style guidelines to which all the built-in Series 5 applications adhere. Several shortcuts are reserved for particular commands and it could be very confusing to the user if these shortcuts are used for other commands in your apps. A style guide is provided with the SDK.

For example, OPL32 supports the following:

- Menu items without shortcuts, by specifying shortcut values between 1 and 31. For these items the value specified is still returned if the item is selected.
- Menu items which are dimmed, or which have checkboxes or option buttons (sometimes known as radio buttons).

  These extra properties are controlled by ORing the following bits together with the shortcut keycode (the constants are found in Const.oph):

```
const KMenuDimmed%                  =$1000   rem Item dimmed
const KMenuCheckBox%                =$0800   rem Check-box
const KMenuOptionStart%             =$0900   rem Option start
const KMenuOptionMiddle%            =$0A00   rem Option middle
const KMenuOptionEnd%               =$0B00   rem Option end
const KMenuSymbolOn%                =$2000   rem Symbol on
const KMenuSymbolIndeterminate%     =$4000   rem Symbol indeterminate
```

The start, middle and end option buttons are for specifying a group of related items that can be selected exclusively (i.e. if one item is selected then the others are deselected). The number of middle option buttons is variable. A single menu card can have more than one set of option buttons and checkboxes, but option buttons in a set should be kept together. For speed, OPL does not check the consistency of these items' specification.

Your application code needs to maintain the state of any checkbox or radio button. Items with KMenuCheckBox% set, has the tick displayed when the menu is presented if KMenuSymbolOn% is also set. The display of the option buttons is similarly controlled by ORing KMenuSymbolOn% with the appropriate shortcut value. The display of ticks and option buttons is automatically changed appropriately when you select one of these items.

As in OPL16, making the whole shortcut value negative will draw a line separator under the menu item, to enable logical grouping of a set of menu items.

Example:

```
mCARD "xxx","yyy",-(KMenuDimmed% OR KMenuCheckBox% OR %A)
```

specifies a dimmed, underlined item "yyy" with an initially unset checkbox and shortcut Control+Shift+A.

A 'Too wide' error is raised if the menu or cascade title length is greater than or equal to 40. Shortcut values must be alphabetic character codes or numbers between the values of 1 and 31. Any other values will raise an 'Invalid arguments' error.

MENU

Usage: Unchanged from OPL16

In OPL16, when the menu is closed and reopened the highlighted item on reopening is the the one last selected, However, this is not the case in OPL32, and hence it is necessary to use `MENU(init%)`, passing back the same variable each time the menu is opened if you wish the menu to reopen with the highlight set on the last selected item.

MINIT...MENU (ERROR HANDLING)

Usage: Unchanged from OPL16

In OPL32, a menu is discarded when an item fails to be added successfully. In effect the previous mINIT statement is discarded together with any previous mCARD statements. This avoids the problem of trying to use a badly constructed menu item.

It is therefore now incorrect to ignore mCARD errors by having an ONERR label around an mCARD call. If you do, the menu is discarded and a 'Structure fault' will then be raised on using mCARD or MENU without first using mINIT again.

It is highly unlikely that any existing code will have done this anyway. The following bad code will not display the menu:

```
mINIT
     ONERR errIgnore1
     mCARD "Xxx","ItemA",0     rem bad shortcut
errIgnore1::
     ONERR errIgnore2
     mCARD "Yyy",""  << 'Structure fault' error (mINIT discarded)
errIgnore2::
ONERR OFF
MENU           << 'Structure fault' again
```

OFF

Usage: `OFF n%`

The minimum time to switch off is now 5 seconds. EPOC32 prevents switchoff if there's an abosolute timer outstanding to go off in less than 5 seconds. In EPOC16 the minimum was 2 seconds.

SCREENINFO

Usage: `SCREENINFO var info%()`

In OPL32 the font ID is a 32-bit integer (see above) and therefore would not fit into a single element of info%(). Hence, the font ID is no longer returned to info%(6), but instead the least significant 16 bits are returned to info%(9) and the most significant 16 bits to info%(10).

TRAP RAISE

Usage: `TRAP RAISE err%`

This clears the TRAP flag and sets ERR value to err%.

GENERAL NEW KEYWORDS

This section provides an alphabetic listing of non-graphics and non-datafile keywords that have been introduced for OPL32. Again, see the separate sections for new graphics and datafile keywords and features.

CAPTION

Usage: `CAPTION caption$,languageCode%`

This specifies an application's public name (or caption) for a particular language and can be used for as many languages as required. (Note that unlike a caption given in the APP keyword, it is enclosed in quotes.) The maximum length of caption$ is 256 characters. If CAPTION is used for any language then the default caption provided in the APP declaration will be discarded. Therfore if the CAPTION keyword is used at all, it is necessary to supply CAPTION statements for all languages in which the application is liable to be used, including the language of the machine on which the application was origianlly developed. CAPTION is only used inside an APP...ENDA construct.

Language codes:

> English 1, French 2, German 3, Spanish 4, Italian 5, Swedish 6, Danish 7, Norwegian 8, Finnish 9, American 10, Swiss-French 11, Swiss-German 12, Portuguese 13, Turkish 14, Icelandic 15, Russian 16, Hungarian 17, Dutch 18, Belgian-Flemish 19, Australian 20, Belgian-French 21, Austrian 22, New Zealand 23, International French 24.

These are defined in Const.oph as KLangEnglish%=1,KLangFrench%=2, etc.

CLEARFLAGS

Usage: `CLEARFLAGS flags&`

CLEARFLAGS clears the flags given in flags& if they have been set by SETFLAGS, returning to the default. See SETFLAGS for the available flags.

CONST

Usage: `CONST  KConstantName = constantValue`

This is used to declare constants which are treated as literals, not stored as data. The declarations must be made outside any procedure, usually at the beginning of the program. KConstantName has the normal type-specification indicators (%, &, $ or nothing for floats). CONST values have global scope, and are not overridden by locals or globals with the same name (in fact the translator will not allow the declaration of locals or globals of the same name). By convention, all constants should be named with a leading 'K' to distinguish them from variables.

It should be noted that it is not possible to define constants with values -32768 (for integers) and -214748648 (for longs) in decimals, but hexadecimal notation may be used instead (i.e. values of $8000 and &80000000 respectively).

The header file Const.oph is released with OPL32. This file may be included in your programs (see INCLUDE) and contains definitions of many standard OPL constants.

DCHECKBOX

Usage: `dCHECKBOX chk%,prompt$`

This command creates a dialog checkbox entry. This is similar to a choice list with two items, except that the list is replaced by a checkbox with the tick either on or off. The state of the checkbox is maintained across calls to the dialog. Initially you should set the live variable chk% to 0 to set the tick symbol off and to any other value for to set it on. chk% is then automatically set to 0 if the box is unchecked or -1 if it is checked when the dialog is closed.

DEDITMULTI

Usage: `dEDITMULTI ptrData&,prompt$,widthInChars%,numberLines%,maxLength%`

This keyword defines a multi-line edit box to go into a dialog. Normally the resulting text would be used in a subsequent dialog, saved to file or printed using the Printer OPX.

Note that the edit box uses any Enter key pressed when it has the focus, so Enter can't be used to exit the dialog, unless another item is provided that can take the focus without using the Enter key. Normal practice is to provide a button that does not use the Enter key to exit a dialog whenever it contains a multi-line edit box. The Enter key is used by a multi-line edit box which has the focus before being offered to any buttons. The Esc key will always cancel a dialog however, even when it contains a multi-line edit box.

ptrData& is the address of a buffer to take the edited data. It could be the address of an array as returned by ADDR, or of a heap cell, as returned by ALLOC (see ADDR and ALLOC). The buffer may not be specified directly as a string and may not be read as such. Instead it should be peeked, byte by byte. The leading 4 bytes at ptrData& contains the initial number of bytes of data following. These bytes are also set by dEDITMULTI to the actual number of bytes edited. For this reason it is convenient to use a long integer array as the buffer, with at least 1+(maxLength%+3)/4 elements. The first element of the array then specifies the initial length.

If an allocated cell is used (probably because more than 64K is required), the first 4 bytes of the cell must be set to the initial length of the data. If this length is not set then an error will be raised. For example if a cell of 100000 bytes is allocated, you would need to poke a zero long integer in the start to specify that there is initially no text in the cell. For example:

```
p&=ALLOC(100000)
POKEL p&,0   rem Text starts at p&+4
```

The following special characters defined in Const.oph may appear in the buffer:

```
CONST KParagraphDelimiter%   =$06
CONST KLineBreak             =$07
CONST KPageBreak             =$08
CONST KTabCharacter          =$09
CONST KNonBreakingTab        =$0a
CONST KNonBreakingHyphen     =$0b
CONST KPotentialHyphen       =$0c
CONST KNonBreakingSpace      =$10
CONST KVisibleSpaceCharacter =$0f
```

Most of the special characters are self-explanatory. A paragraph delimiter marks the end of a paragraph. A line break is a new line which does not signify a new paragraph. Non-breaking characters guarantee that no line break will occur at the point they are inserted. A potential hyphen marks the place where a hyphen may be inserted if the word occurs on the break of a line. A visible space character is the character which is displayed when spaces are set to be visible.

The prompt, prompt$ will be displayed on the left side of the edit box. widthInChars% specifies the width of the edit box within which the text is wrapped, using a notional average character width. The actual number of characters that will fit depends on the character widths, with e.g. more ‘i’s fitting than ‘W’s. numberLines% specifies the number of full lines displayed. Any more lines will be scrolled. maxLength% specifies the length in bytes of the buffer provided, excluding the bytes used to store then length..

The following example presents a three-line edit box which is about 10 characters wide and allows up to 399 characters:

```
CONST KLenBuffer%=399
PROC dEditM:
   LOCAL buffer&(101) REM 101=1+(399+3)/4 in integer arithmetic
   LOCAL pLen&,pText&
   LOCAL i%
   LOCAL c%

   pLen&=ADDR(buffer&(1))
   pText&=ADDR(buffer&(2))
   WHILE 1
     dINIT “Try dEditMulti”
     dEDITMULTI pLen&,”Prompt”,10,3,KLenBuffer%
     dBUTTONS “Done”,%d   REM button needed to exit dialog
     IF DIALOG=0 :BREAK :ENDIF
     PRINT “Length:”;buffer&(1)
     PRINT “Text:”
     i%=0
     WHILE i%<buffer&(1)
       c%=PEEKB(pText&+i%)
       IF c%>=32
         PRINT CHR$(c%);
       ELSE
          PRINT “.”;  REM just print a dot for special characters
       ENDIF
       i%=i%+1
     ENDWH
   ENDWH
ENDP
```

DAYSTODATE

Usage: `DAYSTODATE days&,year%,month%,day%`

This converts days&, the number of days since 1 January 1900, to the corresponding date. This is useful for converting the value set by dDATE, which also gives days since 1 January 1900.

DECLARE EXTERNAL

Usage: `DECLARE EXTERNAL`

This command causes the translator to report an error if any variables or procedures are used before they are declared. It should be used at the beginning of the module to which it applies, before the first procedure. It is useful for detecting 'Undefined externals' errors at translate-time rather than at runtime.

E.g. With DECLARE EXTERNAL commented out, the following translates and raises the error, "Undefined externals, i" at runtime. Adding the declaration causes the error to be detected at translate-time instead.

```
rem DECLARE EXTERNAL
PROC main:
   local i%
   i%=10
   print i
   get
ENDP
```

If you use this declaration, you will need to declare all subsequent variables and procedures used in the module using EXTERNAL as discussed below.

EXTERNAL

Usage: `EXTERNAL variable`

This declares variable as external. This would be required if DECLARE EXTERNAL is specified in the module. E.g.

```
EXTERNAL screenHeight%
```

Usage: `EXTERNAL prototype`

Declares the prototype of a procedure ("prototype" includes the final ":" and the argument list). The procedure may then be referred to before it is defined. This allows parameter type-checking to be performed at translate-time rather than at runtime and also provides the necessary information for the translator to coerce numeric argument types, thus avoiding 'Type violation' errors at runtime. Hence a 'Type violation' error does not result in the following example, even though a "&" does not precede the 2 passed to the procedure two:(),

```
DECLARE EXTERNAL
EXTERNAL two:(long&)

PROC one:
   two:(2)
ENDP

PROC two:(long&)
   ..
ENDP
```

Coercion is reasonable because OPL does not support argument overloading. The same coercion occurs as when calling the built-in keywords.

Following the example of C and C++, you would normally provide a header file declaring all the procedures in the current module and include this header both at the beginning of the module defining these procedures and in any other modules which load this module. Then you should use DECLARE EXTERNAL (see above) at the beginning of the module so that the translator can ensure that these procedures are called with the correct parameter types or types which can be coerced.

An example of usage of DECLARE EXTERNAL and EXTERNAL:

```
DECLARE EXTERNAL
EXTERNAL myProc%:(i%,l&)
REM or INCLUDE "myproc.oph" that defines all your procedures

PROC test:
   local i%,j%,s$(10)

   myProc%:(i%,j%)      REM j% is coerced to a long integer
                        REM as specified by the prototype.
   myProc%:(i%,s$)      REM translator 'Type mismatch' error: string can't
                        REM be coerced to numeric type and vice versa.
   myProc%:(i%)         REM wrong argument count gives tran error
ENDP

PROC myProc%:(i%,l&)    REM Tran checks consistency with prototype above
   ...
ENDP
```

FLAGS

Usage: `FLAGS flags%`

This keyword replaces TYPE for SIBO OPAs. The following values for flags% are defined in Const.oph:

```
CONST KFlagsAppFileBased%=1

CONST KFlagsAppIsHidden%=2
```

KFlagsAppFileBased% is used if your application can create files. It will then be included in the list of applications offered when the user creates a new file from the System screen.

KFlagsAppIsHidden% prevents the application from appearing in the Extras bar. It is very unusual to have this flag set.

GETDOC$

Usage: `docname$=GETDOC$`

This returns the name of the current document. See also SETDOC, and above for explanation of Series 5 "documents".

GETEVENT32

Usage: `GETEVENT32 ev&()`

GETEVENT32 ev&() gets all event types handled by GETEVENT ev%() and additionally pointer (pen or mouse) events. The latter are too large to fit into the array of integers provided for GETEVENT ev%(), so this separate funcion, GETEVENT32 ev&(), has been provided. ev&() must have at least 16 elements.

All events return a 32-bit time stamp. The Window ID mentioned below refers to the value returned by the gCREATE keyword.

The modifier values are given in “Modification of character codes” above. Scancode values for a keypress specify a location on the keyboard. (See the appendix for the values of scancodes.)

GETEVENT32 returns more information than GETEVENT, as listed below:

If (ev&(1) AND &400) = 0 then the event is a keypress. In this case, ev&(1) = keycode, ev&(2) = time stamp, ev&(3) = scan code, ev&(4) = modifier, ev&(5) = repeat. Note that this is strictly the repeat value, i.e. if there is only one keypress, then the value of ev&(5) is 0.

For all the other event types, ev&(1) is greater than &400 (i.e. (ev&(1)≠&400):

ev&(1)=&401 is ‘Focus gained’, ev&(2) = time stamp.

ev&(1)=&402 is ‘Focus lost’, ev&(2) = time stamp.

ev&(1)=&403 is ‘Machine switched on’, ev&(2) = time stamp. Note that this event is not enabled unless the appropritate flag is set (by default it is not): see SETFLAGS.

ev&(1)=&404 is ‘Terminated’. If this event occurs GETCMD$ should be called to find out what action should be taken - see the full Programming Manual for more details.

ev&(1)=&406 is ‘key down’, ev&(2) = time stamp, ev&(3) = scan code, ev&(4) = modifiers.

ev&(1)=&407 is ‘key up’, ev&(2) = time stamp, ev&(3) = scan code, ev&(4) = modifiers.

ev&(1)=&408 is ‘pointer event’, ev&(2)=time stamp, ev&(3)=window ID , ev&(4)=pointer type (see below), ev&(5) = modifiers, ev&(6) = x-coordinate, ev&(7) = y-coordinate, ev&(8) = x-coordinate relative to parent window, ev&(9) = y-coordinate relative to parent window.

For ‘pointer event’ type ,ev&(4), gives one of the following values:

0 = pen or button1 down

1 = pen or button1 up

2 = button2 down

3 = button2 up

4 = button3 down

5 = button3 up

6 = drag

7 = move

On the PC, button 1 corresponds to the left mouse button, button 2 to the middle mouse button (for a mouse with three buttons only) and button 3 to the right mouse button. Currently only types 0,1 and 6 are supported on Series 5, but other pointer event types may be supported on future machines. The values which are applicable to both the PC and Series 5 are given in Const.oph as KEvPtrPenDown&, with

KEvPtrButton1Down&=KEvPtrPenDown&, and KEvPtrPenUp& with KEvPtrButton1Up&=KEvPtrPenUp&.
ev&(1)=&409 is 'pointer enter', ev&(2) = time stamp, ev&(3) = window ID.
ev&(1)=&40A is 'pointer exit', ev&(2) = time stamp, ev&(3) = window ID.

Some pointer events, and pointer enters and exits, can be filtered out to avoid being swamped by unwanted event types. See POINTERFILTER for details.

Note: for other unknown events, ev&(1) contains &1400 ORed with the code returned by the window server. ev&(2)=timestamp, ev&(3)=window ID, followed by the rest of the data returned by the window server in ev&(4), ev&(5), etc.

GETEVENTA32

Usage: `GETEVENTA32 status%,ev&()`

Asynchronous version of GETEVENT32.

GETEVENTC

Usage: `GETEVENTC(status%)`

Cancels the previously called GETEVENTA32 function with status stat%. Note that GETEVENTC consumes the signal (unlike IOCANCEL), so IOWAITSTAT should not be used after GETEVENTC.

INCLUDE

Usage: `INCLUDE file$`

Includes a file, file$, which may contain CONST definitions, prototypes for OPX (OPL language extension DLLs) procedures and prototypes for module procedures (see EXTERNAL). The included file may *not* include module procedures themselves. Procedure and OPX procedure prototypes allow the translator to check parameters and coerce numeric parameters (that are not passed by reference) to the required type. INCLUDing a file is logically identical to replacing the INCLUDE statement by the file's contents.

The filename of the header may or may not include a path. If it does include a path, then OPL will only scan the specified folder for the file. However, the default path for INLCUDE is \System\Opl\, so when INCLUDE is called *without specifying a path*, OPL looks for the file firstly in the current folder and then in \System\Opl\ in all drives from Y: to A: and then in Z:, excluding any remote drives.

IOWAITSTAT32

Usage: `IOWAITSTAT32 var stat&`

This is provided in OPL32 to take a 32-bit status word. IOWAITSTAT32 should be called only when you need to wait for completion of a request made using a 32-bit status word when calling an asynchronous OPX procedure.

Important note: The initial value of a 32-bit status word while it is still pending (i.e. waiting to complete is &80000001 (KStatusPending32& in Const.oph). For a 16-bit status word the 'pending value' is -46 (KErrFilePending%).

MCASC

Usage: `mCASC title$,item1$,hotkey1%,item2$,hotkey2%`

This creates a cascade for a menu, on which less important menu items can be displayed. The cascade must be defined before use in a menu card. The following is an example of a 'Bitmap' cascade under the File menu of a possible OPL drawing application.

Definition:

```
mCASC "Bitmap","Load",%L,"Merge",%M
```

Insertion into File menu,

```
mCARD "File","New",%n,"Open",%o,"Save",%s,"Bitmap>",16,"Exit",%e
```

The trailing ">" character specifies that a previously defined cascade item is to be used in the menu at this point: is not itself displayed in the menu item. A cascade has a filled arrow head displayed along side it in the menu. The cascade title in mCASC is also used only for identification purposes and is not displayed in the cascade itself. This title needs to be identical to the menu item text apart from the '>'. For efficiency, OPL doesn't check that a defined cascade has been used in a menu and an unused cascade will simply be ignored. To display a '>' in a cascaded menu item, you can use '>>'.

Shortcut keys used in cascades may be ORed with the appropriate constant values (see mCARD) to enable checkboxes, option buttons and dimming of cascade items.

As is typical for cascade titles, a shortcut value of 16 is used in the example above. This prevents the display or specification of any shortcut character. However, it is possible to define a shortcut value for a cascade title if required, for example to cycle through the options available in a cascade.

MPOPUP

Usage:`mPOPUP(x%,y%,posType%,item$,hotKey%,itemx$,hotKeyx%,...)`

Presents a popup menu. mPOPUP returns the value of the keypress used to exit the popup menu, this being 0 if Esc is pressed. Note that mPOPUP defines and draws a menu itself and is *not* used in conjunction with mINIT and MENU as other menu commands are.

posType% is the position type controlling which corner of the popup menu x%,y% specifies and can take the values,

0 top left corner

1 top right corner

2 bottom left corner

3 bottom right corner

item$ and hotKey% can take the same values as for mCARD, with hotKey% taking the same constant values to specify checkboxes etc. Note, however, that cascades in popup menus are not supported.

E.g.

```
mPOPUP (0,0,0,"Continue",%c,"Exit",%e)
```

specifies a popup menu with 0,0 as its top lefthand corner with the items "Continue" and "Exit", with the shortcuts Ctrl+C and Ctrl+E respectively.

POINTERFILTER

Usage: `POINTERFILTER filter%,mask%`

This allows pointer events in the current window to be filtered out or back in. OR the following flags together to achieve the desired filter% and mask%:

None	$0
Enter/Exit	$1
Move	$2
Drag	$4

These constants can be found in Const.oph as KPointerFilterEnterExit%, KPointerFilterMove% etc.

The bits set in filter% specify the settings to be used, 1 to filter out the event and 0 to remove the filter. Only those bits set in mask% will be used for filtering. This allows the current setting of a particular bit to be left unchanged if that bit is zero in the mask. (i.e. mask% dictates what to change and filter% specifies the setting to which it should be changed). Initially the events are not filtered out.

Note that the 'move' event, which is only relevant to mouse pointers, needs to be activated by adding the parameter 'Pointer 1' to the 'wsini.ini' files.

SETDOC

Usage: `SETDOC cmd$(2)`

`CREATE cmd$(2),a,a$,b$` or `gSAVEBIT cmd$(2)`

As discussed in the section 'System-level changes', a document is a file that is recognised by the System screen. When a document is selected from the System screen, its associated application is launched. All OPL file types can be created as documents, provided the rules described below are followed.

SETDOC should be called just before the creation of the file that is to become the document.

name$ should be exactly the same as the filename passed to CREATE or gSAVEBIT otherwise a non-document file will be created. Example of document creation:

```
SETDOC "myfile"
CREATE "myfile",a,a$,b$ REM creates document rather than external file
```

SETDOC should also be called when opening a document to allow the System screen to display the correct document name in its task list.

Database documents, created using CREATE, and multi-bitmap documents, created using gSAVEBIT, will automatically be contain your application UID in the file header. For binary and text file documents created using IOOPEN and LOPEN, it is the programmer's responsibility to save the appropriate header in the file. This is a fairly straight-forward process and the following suggests one way of finding out what the header should be:

Create a database or bitmap document in a test run of you application using SETDOC as shown above.

Use a hex editor or hex dump program to find the 1st 16 bytes, or run the program below which reads the four long integer UIDs from the test document.

Write these four long integers to the start of the file you create using IOOPEN.

```
DECLARE EXTERNAL
CONST KIoOpenModeOpen%=$0000
CONST KIoOpenFormatBinary%=$0000

EXTERNAL readUids:(file$)

PROC main:
   LOCAL f$(255)
   WHILE 1
      dINIT “Show UIDs in document header”
      dPOSITION 1,0
      dFile f$,”Document,Folder,Drive”,0
      IF DIALOG=0
         RETURN
      ENDIF
      readUids:(f$)
   ENDWH
ENDP

PROC readUids:(file$)
   LOCAL ret%,h%
   LOCAL uid&(4),i%

   ret%=IOOPEN(h%,file$, KIoOpenModeOpen% OR KIoOpenFormatBinary%)
   IF ret%>=0
      ret%=IOREAD(h%,ADDR(uid&()),16)
      PRINT “Reading “;file$
      IF ret%=16
         WHILE i%<4
            i%=i%+1
            print “  Uid”+num$(i%,1)+”=”,hex$(uid&(i%))
         ENDWH
      ELSE
         PRINT “  Error reading: “;
         IF ret%<0
            PRINT err$(ret%)
         ELSE
            PRINT “Read “;ret%;” bytes only (4 long integers required)”
         ENDIF
      ENDIF
      IOCLOSE(h%)
   ELSE
      PRINT “Error opening: “;ERR$(ret%)
   ENDIF
ENDP
```

Creating text file documents using IOOPEN or LOPEN has two special requirements:

1. You will need to save the required header as the first text record. This will insert the standard text file line delimiters CR LF (hex 0D 0A) at the end of the record.

2. The specific 16 bytes required for your application may itself however contain CR LF. Since you should know when this is the case, you will need to read records until you have reached byte 16 in the document. This is clearly not a desirable state of affairs but is inescapable given that text files were not designed to have headers. It is recommended that you request a new UID for your application if it contains CR LF.

See also GETDOC$.

SETFLAGS

Usage:`SETFLAGS flags&`

SETFLAGS sets flags to produce various effects when running programs. Use CLEARFLAGS to clear the flags.

The following constant values for flags& are supplied in Const.oph:

```
const KRestrictTo64K&=&0001
const KAutoCompact&=&0002
const KTwoDigitExponent&=&0004
const KSendSwitchOnMessage&=&010000
```

KRestrictTo64K& restricts the memory available to your application to 64K, emulating OPL16 on Sibo. See the section ‘32-bit addressing’ for details and for the reason it has been provided.

KAutoCompact& enables auto-compaction on closing databases.

KTwoDigitExponent& tells OPL32 to raise an error for values greater than or equal to 1.0E+100 in magnitude, instead of allowing 3-digit exponents (for compatibilty with OPL16).

With KSendSwitchOnMessage& set, GETEVENT32 returns event code KEvSwitchOn& ($403) when the machine switches on. By default this event is not enabled.

DATABASE OVERVIEW: THE NEW MODEL

OPL32 uses the relational database management system (DBMS) of EPOC32 which supports SQL (Standard Query Language). Apart from the removed keywords RECSIZE, COMPRESS and ODBINFO, the OPL16 methods of database programming are completely understood by the OPL32 model and existing code will not need to change.

DATABASES, TABLES, VIEWS, FIELDS AND FIELD HANDLES

To describe the new model it is necessary to expand upon the terminology that is currently used in the OPL16 manual.

A SIBO datafile corresponds more or less to a single ‘table’ in a DBMS database. A database can contain one or more tables. A table, like a datafile in SIBO, contains records which are made up of fields. Unlike OPL16, the field names as well as the table names are stored in the database.

With the OPL16 statement

```
CREATE “datafile”,A,f1%,f2%
```

OPL32 would create a database called “datafile” and a table with the default name “Table1” would be added to it. The field names are derived from the f1% and f2% which are now called ‘field handles’. The type of the field, as always, is defined by these handles.

With OPL32 it is also possible to use, for example,

```
CREATE “people FIELDS name, number TO phoneBook”,A,n$,number$
```

This will create a table called “phoneBook” in the database called “people”, creating the database too if it does not exist. The table will have fields “name” and “number”, whose respective types are specified by the field handles n$ and number$ both strings in this example. (Note that CREATE creates a table.) An error is raised if the table already exists in the database. DBMS does not allow the database to be open when a table (or an index: see the Database OPX section in the OPX Overview chapter) is created in it. You should first close the database, i.e. close any tables previously opened in it, before using CREATE.

With the OPL16 OPEN statement,

```
OPEN “datafile”,A,f1%,f2%
```

OPL32 would open the default table “Table1” and provide access to as many fields as there are handles supplied.

In OPL32 it is also possible to open multiple ‘views’ on a table and to specify which fields are to be available in a view. E.g.

```
OPEN “people SELECT name FROM phoneBook”,A,n$
```

This view gives you access to just the “name” field from the “phoneBook” table.

The string from ‘SELECT’ onwards in the OPEN statement forms an SQL query which is passed straight on to the underlying EPOC32 DBMS. The SQL command-set is specified in an Appendix at the end of this manual.

A more advanced view, ordered by an index (described later), would be opened as follows,

```
OPEN “people SELECT name, number FROM phoneBook ORDER BY name ASC, number
DESC”,A,n$,num%
```

This would open a view with “name” fields in ascending alphabetical order and if any names were the same then the number field would be used to order these records in descending numerical order.

Note that, to avoid ambiguity, database names with embedded spaces must have enclosing quotes around them, using the standard OPL syntax for embedding quotes inside strings e.g.:

```
CREATE “””clients with spaces”” FIELDS name, tel TO phone”, D, n$, t$
OPEN  “””clients with spaces”” SELECT name, tel FROM phone”, D, n$, t$
CREATE “””clients with spaces”””, D, n$, t$
OPEN  “””clients with spaces”””, D, n$, t$
```

TRANSACTIONS

‘Transactions’ allow changes to a database to be committed in stages. It is necessary to use transactions in database operations to achieve reasonable speeds.

Transactions are a truly fundamental part of the DBMS model, so much so that without the use of transactions you will find that writing to a DBMS database is in fact slower than the equivalent operations in OPL16. With transactions however, OPL32 database handling is far faster than that of OPL16.

See BEGINTRANS, COMMITTRANS, INTRANS and ROLLBACK.

RECORD POSITION

In the DBMS model, as with most modern relational database models, absolute record position does not have much significance.

‘Bookmarks’ can be assigned to particular records to provide fast record access and should be used in preference to POS and POSITION when opening views using the new OPL32 “OPEN...SELECT...” or “CREATE ... FIELDS...” statements. POS and POSITION can be used safely on tables opened or created using an OPL16-style OPEN or CREATE statement. Bookmarks should not, however, be used in conjunction with the POS and POSITION keywords as it can cause these keywords, kept mainly for SIBO compatibility, to become inaccurate. See BOOKMARK, KILLMARK, GOTOMARK, POS and POSITION.

When using the OPL32 extensions to CREATE and OPEN, you should also usually use the new INSERT, MODIFY, PUT and CANCEL keywords in preference to the APPEND and UPDATE OPL16 keywords. APPEND and UPDATE still work as expected, but do not naturally fit in the DBMS model. MODIFY allows records to be changed without being moved to the end of the set (as UPDATE still does). Instead of copying the current record to the end of the set as APPEND does, INSERT appends a new record to the end of the set with numeric fields set to 0 and string fields empty if values have not been assigned to them.

PUT marks the end of a database’s INSERT or MODIFY phase and makes the changes permanent. CANCEL marks the end of a database’s INSERT or MODIFY phase and discards the changes made during that phase.

LOGICAL FILENAMES

The range for logical filenames, now also known as the logical view name, has been expanded from A-D to A-Z.

INDEXES

Indexes can be constructed on a table using several fields as keys. These indexes are subsequently used to provide major speed improvements when opening a table or views on them.

Further database functionality is provided in the DBASE OPX, discussed in the OPX section below.

OPENING A DATABASE CREATED BY THE DATA APPLICATION

You can open a file created by the Data application files in an OPL program. The file is opened for reading only because if it were written to, OPL would have to discard all the formatting characters. An OPL program can create a new OPL database and copy the Data application records into it if necessary.

To open a Data application database that has one string field which you need to access, you could use:

```
OPEN "file",a,a$
```

Types not supported by OPL will be ignored. Note that integer fields in the Data application correspond to long integer fields in OPL: the Data application does not support (16-bit) integer fields. The types and order of the OPL field handles must match the fields in the Data file. For example, if the Data app file "Data2" contains:

1. long integer field
2. datetime field (ignored by OPL)
3. string field
4. float field

you could access the fields supported by OPL using:

```
OPEN "Data2",A, f1&,f2$,f3
```

It would be better, however, to use the SQL SELECT clause to name the required Data file fields explicitly. For this to be possible it is necessary to use table name and the same field names as are used by Data. All Data files have a single table called Table1. The fields (referred to internally as columns in Data) are named "ColA1", "ColA2", etc.

So, with the field types from the previous example, the Data file could be opened using:

```
OPEN "Data2 SELECT ColA1,ColA3,ColA4 FROM Table1",a,f1&,f2$,f3
```

DATABASE HANDLING - CHANGED KEYWORDS

CLOSE

Usage: `CLOSE`

This command closes the current view on a database. If there are no other views open on the database then the database itself will be closed.

COUNT

Usage: `COUNT`

If you try to count the number of records in a rowset while updating the rowset an 'Incompatible update mode' error (-125) will now be raised (This will occur between assignment and APPEND / UPDATE or between MODIFY / INSERT and PUT).

CREATE

Usage: `CREATE  tableSpec$,log,f1,f2,...`

This creates a table in a database. The database is also created if necessary. Immediately after calling CREATE, the file and view (or table) is open and ready for access.

tableSpec$ contains the database filename and optionally a table name and the field names to be created within that table. For example:

```
CREATE "clients FIELDS name(40), tel TO phone", D, n$, t$
```

The filename is "clients". The table to be created within the file is "phone". The comma-separated list, between the keywords 'FIELDS' and 'TO', specifies the field names whose types are specified by the field handles (i.e. n$, t$).

The "name" field has a length of 40 bytes, as specified within the brackets that follow it. The "tel" field has the default length of 255 bytes. This mechanism is necessary for creating some indexes. See the section on the Database OPX for details of index creation.

The logical view name log can be any letter in the range A to Z and is used to identify the view to other database commands such as USE.

COMPATIBILITY WITH OPL16

As in OPL16, the table specification may contain just the filename. In this case the table name will default to "Table1" and the field names will be derived from the handles: "$" replaced by "s", "%" by "i", and "&" by "a". E.g. n$ becomes ns. Knowing this allow views to be opened on tables (called Table1) that were created with the OPL16 method. However, it would be better to create the fields with proper names in the first place. For example:

```
CREATE "clients",A,n$,t%,d&
```

is a short version of

```
CREATE "clients FIELDS ns,ti,da TO Table1",A,n$,t%,d&
```

both creating "Table1". Database "clients" is also created if it does not yet exist.

OPEN

Usage: `OPEN query$,log,f1,f2`

This command opens an existing table (or a 'view' of a table) from an existing database, giving it the logical view name log and handles for the fields f1, f2. log can be any letter in the range A to Z.

query$ specifies the database file, the required table and fields to be selected from that file.

For example:

```
OPEN "clients SELECT name, tel FROM phone",D,n$,t$
```

The file name here is "clients" and the table name is "phone". The field names are enclosed by the keywords SELECT and FROM and their types should correspond with the list of handles (i.e. n$ indicates that the "name" field is a string).

Replacing the list of field names with * selects all the fields from the table.

query$ is also used to specify an ordered view and if a suitable index has been created, then it will be used. See Database OPX for details. For example,

```
OPEN "people SELECT name,number FROM phoneBook ORDER BY name ASC,number
DESC",G,n$,num%'
```

would open a view with 'name' fields in ascending alphabetical order and if any names were the same then the number field would be used to order these records in descending numerical order.

COMPATIBILITY WITH OPL16

As in OPL16, the query may contain just the filename. In this case a table with the default name "Table1" would be opened if it exists. The field names would then be unimportant as access will be given to as many fields as there are supplied handles. The type indicators on the field handles must match the types of the fields.

POS

Usage: `p%=POS`

This returns the number of the current record in the current view. The value returned by POS can become inaccurate if used in conjunction with bookmarks and multiple views on a table. Accuracy can be restored by using FIRST or LAST on the current view.

POSITION

Usage: `POSITION x%`

Makes record number x% the current record in the current view. By using bookmarks (see BOOKMARK) and editing the same table via different views, positional accuracy can be lost and POSITION x% could access the wrong record. Accuracy can be restored by using FIRST or LAST on the current view.

POS and POSITION still exist mainly for OPL16 compatibility reasons and you should avoid using them in conjunction with bookmarks and editing of the same table via separate views.

DATABASE - NEW KEYWORDS

DELETE

Usage: `DELETE dbase$, table$`

This deletes the table, table$, from the database, dbase$. To do this all views of the database, and hence the database itself, must be closed.

INSERT

Usage: `INSERT`

This inserts a new, blank record into the current view of a database. The fields can then be assigned to before using PUT or CANCEL.

MODIFY

Usage: `MODIFY`

This allows the current record of a view to be modified without moving the record. The fields can then be assigned to before using PUT or CANCEL.

PUT

Usage: `PUT`

This marks the end of a database's INSERT or MODIFY phase and makes the changes permanent.

CANCEL

Usage: `CANCEL`

This marks the end of a database's INSERT or MODIFY phase and discards the changes made during that phase.

BOOKMARK

Usage: `b%=BOOKMARK`

This puts a bookmark at the current record of the current database view. The value returned can be used in GOTOMARK to make the record current again. Use KILLMARK to delete the bookmark.

GOTOMARK

Usage: `GOTOMARK b%`

This makes the record with bookmark b%, as returned by BOOKMARK, the current record. b% must be a bookmark in the current view.

KILLMARK

Usage: `KILLMARK b%`

This removes the bookmark b%, which has previously been returned by BOOKMARK, from the current view of a database.

BEGINTRANS

Usage: `BEGINTRANS`

This begins a transaction on the current database. The purpose of using a transaction is to allow changes to a database to be committed in stages. Once a transaction has been started on a view (or table) then all database keywords will function as usual, but the changes to that view will not be made until COMMITTRANS is used.

See also COMMITTRANS, ROLLBACK, INTRANS.

COMMITTRANS

Usage: `COMMITTRANS`

This commits the transaction on the current view.

See also BEGINTRANS, ROLLBACK, INTRANS.

INTRANS

Usage: `i&=INTRANS`

Finds out whether the current view is in a transaction. Returns -1 if in a transaction or 0 if not.

See also BEGINTRANS.

ROLLBACK

Usage: `ROLLBACK`

Cancels the current transaction on the current view. Changes made to the database with respect to this particular view since BEGINTRANS was called, will be discarded.

See also BEGINTRANS, COMMITTRANS.

COMPACT

Usage: `COMPACT file$`

This compacts the database file$, rewriting the file in place. All views on the database and the hence the file itself should be closed before calling this command. This should not be done to often since it uses considerable processor power.

Compaction can also be done automatically on closing a file (see SETFLAGS).

GRAPHICS OVERVIEW

UP TO 64 DRAWABLES SUPPORTED

OPL32 supports up to 64 open windows and/or bitmaps at any time.

COLOUR MODES

OPL32 supports the various colour modes available on the Series 5:

- 2 colour mode (black and white). This is stored as one bit per pixel (bpp).
- 4 colour mode (white, light grey, dark grey and black). This is stored as 2bpp.
- 16 colour mode (white, 14 greys and black). This is stored as 4bpp.

A window's colour mode is set using gCREATE.

N.B. Black and white mode is in fact implemented by mapping dark grey to black and light grey to white.

N.B. 16-colour mode uses twice the memory and much more power than 4-colour mode, and the hardware switches to 16-colour mode if any window displayed has this mode.

The display power consumption is dependent on both the colour mode and the pattern on the display, as follows:

- Colour mode: Power consumption doubles from 1bpp to 2bpp and again from 2bpp to 4bpp.
- Pattern: The worst power consumption for the display is produced by a checker board pattern. Grey areas also increase the power consumption considerably.

Overall the current taken by the display is between 25% and 60% of the total idle current:

- 25%: 2bpp plain screen (e.g. plain Word screen).
- 40%: 2bpp grey areas on screen (e.g. Calculator screen).
- 60%: 4bpp bit map on the screen.

It is difficult to be more precise since the power consumption is very dependent on what is being displayed.

NO CONCEPT OF A GREY PLANE: GREY IS JUST ONE OF THE COLOURS

There is no concept of a grey plane, gGREY just draws grey. In addition there is gCOLOR which takes red, green and blue arguments which are mapped to black, white, or a shade of grey for non-colour screens.

BLACK AND WHITE BITMAPS DIFFER FROM BLACK AND WHITE WINDOWS

Drawing colours to a black and white (2-colour mode) bitmap created using gCREATEBIT does not produce the same results as drawing the same colours to a black and white window.

In particular, if you draw in 16 colours to a black and white bitmap then dithering occurs, whereas if you draw exactly the same straight to a 2-colour graphics window you just get black and white.

This enables grey printing on black and white printers.

N.B. It is recommended that you use many small windows rather than a few huge ones (providing there is not too much overlap) because all OPL windows have a backup bitmap to avoid redraw messages. So when using *n* bpp, a full-screen window's backup bitmap requires

$$\frac{n \times 640 \times 240}{8} bytes = n \times 19200 bytes$$

A further benefit is that the file size will be smaller if the bitmap is saved, with just one bit per pixel (1bpp) used for black and white bitmaps.

GRAPHICS - CHANGED KEYWORDS

DEFAULTWIN

Usage: `DEFAULTWIN mode%`

The default window, window ID 1, uses 4-colour mode initially. DEFAULTWIN 1 therefore just clears the screen, leaving the window in 4-colour mode. Clearing of the screen ensures OPL16 compatibility: on SIBO the old window had to be deleted and a new one created to change the colour mode. mode% of 0 changes the screen to 2-colour mode (actually results in a mapping of greys to white or black) and mode% of 2 changes to 16-colour mode, as expected. (Using DEFAULTWIN with either of these values also clears the screen.)

GBORDER

Usage: `gBORDER flags% [,w%,h%]`

If flags% is 2 or 4, then only a border is drawn without any gap around the drawable. However, initially setting a border with flags% of 1 or 3 and then changing it to 2 or 4 respectively removes the shadow, as in OPL16. All borders are drawn in black.

See gXBORDER for more border styles (for SIBO type borders and OPL32 borders).

GBUTTON

Usage: `gBUTTON text$,type%,width%,height%,state%`

```
gBUTTON text$,type%,width%,height%,state%,bitmapId&,maskId&
gBUTTON text$,type%,width%,height%,state%,bitmapId&,maskId&,layout%
```

type%=2 specifies the Series 5. The state% values are the same as for the Series 3a (type%=1).

There is added support for bitmaps in gBUTTON. Three extra optional arguments can be passed which give the bitmap ID, the mask ID and the layout for the button respectively. maskId% can be 0 to specify no mask.

The following constants, defined in Const.oph, should be used for layout% to specify relative positions of the text and icon on a button,

```
const KButtTextRight%   =0
const KButtTextBottom%  =1
const KButtTextTop%     =2
const KButtTextLeft%    =3
```

These values can also be ORed with the following constants to specify how a button's excess space is to be allocated,

```
const KButtExcessShare%      =$00
const KButtExcessToText%     =$10
const KButtExcessToPicture%  =$20
```

When the layout is such that the text is at the top or the bottom, then text and picture are centred vertically and horizontally in the space allotted to them. If the layout has text to the left or right, then the text is left aligned in the space allotted to it and the picture is right or left aligned respectively. Both text and picture are centred vertically in this case.

Examples:

layout%=$13 will create a button with text on the left and left aligned in any excess space.

layout%=$20 will create a button with text on the right and the picture left aligned in any excess space.

For standard toolbar buttons you would use layout%=$10 to put the text on the right.

For a picture only with no text use text$="".

Normal OPL bitmaps can be used, but 'Invalid arguments' error is raised if you use OPL windows for gBUTTON. Read-only bitmaps may also be loaded using OPX functions declared in BMP.OXH.

GCLOCK

Usage:`gCLOCK ON/OFF [,mode%,offset&,format$,font&,style%]`

This does not support mode% 1-5 because these modes were used only for Series3 Classic compatibility on the Series3a. Also, gCLOCK 10 has been replaced by the more complete gCLOCK 11 where the contents of the format string is as specified below:

%% Include a single '%' character in the string

%* Abbreviate following item. (The asterisk should be inserted between the % and the number or letter, e.g. %*1). In most cases this amounts to omitting any leading zeros, for example if it is the first of the month "%F %*M"will display as "1" rather than "01".

%:*n* Interpret the argument as a system time separator character. *n* is an integer between zero and three inclusive which indicates which time separator character is to be used. For European time settings, only n=1 and n=2 are used, giving the hours/minutes spearator and minutes/seconds separator respectively.

%/*n* Interpret the argument as a system date separator character. *n* is an integer between zero and three inclusive which indicates which date separator character is to be used. For European time settings, only n=1 and n=2 are used, giving the day/month spearator and month/year separator respectively.

%1 Interpret the argument as the first component of a three component date (ie a date including day, month and year) where the order of the components is determined by the system settings. The possibilities are: dd/mm/yyyy, (European), mm/dd/yyyy (American), yyyy/mm/dd (Japanese).

%2 Interpret the argument as the second component of a three component date where the order has been determined by the system settings (see %1).

%3 Interpret the argument as the third component of a three component date where the order has been determined by the system settings (see %1).

%4 Interpret the argument as the first component of a two component date (ie a date including day and month only) where the order has been determined by system settings.

%5 Interpret the argument as the second component of a two component date where the order has been determined by the system settings.

%A Interpret the argument as text corresponding to am or pm according to the current language and time of day. Text is printed even if 24 hour clock is in use. Text may be specified to be printed before or after the time, and a trailing or leading space as appropriate will be added. The abbreviated version ("%*A") removes this space. Optionally, a minus or plus sign may be inserted between the % and the A. This operates as follows: "%-A" positioned before the time in the format string causes am/pm text to be inserted only if the system setting of the am/pm symbol position is set to display before the time. No am/pm text may be inserted after the time if the - is inserted. Similarly, "%+A" positioned after the time causes am/pm text to be inserted only if the system setting of the am/pm symbol is set to display after the time. No am/pm text will be inserted before the time if a + is inserted in the string. "%+A" and "%-A" cannot be abbreviated.

%B As %A, except that the am/pm text is only inserted if the system clock setting is 12 hour. (This should be used in conjunction with %J.)

%D Interpret the argument as the two-digit day number in month (in conjunction with %1 etc).

%E Interpret the argument as the day name. Abbreviation is language specific (3 letters in English).

%F Use this at the beginning of a format string to make the date/time formatting locale independent. This fixes the order of the following day/month/year component(s) in their given order, removing the need to use %1 to %5, allowing individual components of the date to be printed. (No abbreviation.)

%H Interpret the argument as the two-digit hour component of the time in 24 hour clock format.

%I Interpret the argument as the two-digit hour component of the time in 12 hour clock format. Any leading zero is automatically suppressed, regardless of whether an asterisk is inserted or suppressed in abbreviated form.

%J Interpret the argument as the two-digit hour component of time in either 12 or 24 hour clock format depending on the corresponding system setting. When the clock has been set to 12 hour format, the hour's leading zero is automatically suppressed regardless of whether an asterisk has been inserted between the % and J.

%M Interpret the argument as the two-digit month number (in conjunction with %1 etc).

%N Interpret the argument as the month name (in conjunction with %1 etc). When using non-fixed (locale) format this causes all months following %N in the string to be written in words. When using fixed format (%F) %N may be used alone to insert a month name. Abbreviation is language specific (3 letters in English).

%S Interpret the argument as the two-digit second component of the time.

%T Interpret the argument as the two-digit minute component of the time.

%W Interpret the argument as the two-digit week number in year, counting the first (part) week as week 1.

%X Interpret the argument as the date suffix. When using non-fixed (locale) format, this causes a suffix to be put on any date following %X in the string. When using fixed format (%F), %X following any date appends a suffix for that particular date. Cannot be abbreviated.

%Y Interpret the argument as the four digit year number (in conjunction with %1 etc). The abbreviation is the last two digits of the year.

%Z Interpret the argument as the three digit day number in year.

Some examples of the use of these format strings are as follows. The example use is 1:30:05 pm on Wednesday, 1st January 1997, with the locale setting of European dates and with am/pm after the time:

1. "%-A%I:%T:%S%+A" will print the time in 12 hour clock, including seconds, with the am/pm either inserted before or after the time, depending on the system setting. So the example time would appear as, "1:30:05 pm".

2. "%F%E %*D%X %N %Y" will print the day of the week followed by the date with suffix, the month as a word and the year. For example, "Wednesday 1st January 1997".

3. "%E %D%X%N%Y %1 %2 %3" will use the locale setting for ordering the elements of the date, but will use a suffix on the day and the month in words. For example, "Wednesday 01st January 1997".

4. "%*E %*D%X%*N%*Y %1 %2 '%3" will be similar to 3., but will abbreviate the day of the week, the day, the month and the year, so the example becomes "Wed 1st Jan 97".

5. "%M%Y%D%1%/0%2%/0%3" will appear as "01/01/1997". This demonstrates that the ordering of the %D, %M and %Y is irrelevant when using locale-dependent formatting. Instead the ordering of the date components is determined by the order of the %1, %2, and %3 formatting commands.

style% may take any of the values used to specify gSTYLE, other than 2 (underlined).

The values $10, $20, $40 and $80 which in OPL16 could be ORed with mode% to produce extra feature are no longer supported as they are no longer supported by the system. However, there are additional features of the basic clocks which partially replace these. The digital clock (mode%=8) automatically displays the day of the week and day of the month below the time. The extra large analog clock (mode%=9) automatically displays a second hand.

A new flag, which has the value $100, may, however, be ORed with mode% so that the offset may be specified in seconds rather than minutes. The offset is a long in OPL32 to enable a whole day to be specified when the offset is in seconds.

A note should also be made that a "General Failure" error will result if you attempt to use an invalid format. Invalid formats inlcude using %: and %/ followed by 0 or 3 when in European locale setting (when these separators are without meaning) and using %+ and %- followed by characters other than A or B.

GCREATE

Usage: `gCREATE(x%,y%,width%,height%,vis%,flags%)`

flags% (instead of grey%) specifies the graphics mode to use and shadowing on the window. By default the graphics mode is 2-colour and there is no shadow.

The least significant 4 bits of flags% gives the colour-mode as before 0 (2 colour-mode), 1 (4 colour-mode), 2 (16 colour-mode). The next 4 bits may be set to specify the shadowing on the window. If 0, the window has no shadow. The next 4 bits give the shadow height relative to the window behind it (a height of *N* units gives a shadow of *N*´2 pixels, but this may change on higher resolution machines).

The flags% argument is most easily specified in hexadecimal:

flags%=$412 specifies a 16 colour-mode ($2), shadowed window ($1), with height 4 units ($4)

above the previous window.

flags%=$010 specifies a black and white shadowed window at the same height as the previous window.

flags%=$101 specifies a 4 colour mode window with no shadow (height ignored if shadow disabled).

flags%=$111 specifies a 4 colour mode window with shadow of 2 units above the window behind.

Note again that 64 drawables (including the default window) may be open at any time, although it is recommended that you use as few windows as possible at any one time. Eight would be a sensible maximum number of windows in practice, although bitmaps may also be used in addition to windows.

GCREATEBIT

Usage: `gCREATEBIT(w%,h%)`

`gCREATEBIT(w%,h%,mode%)`

gCREATEBIT may be used with an optional third parameter which specifies the graphics mode of the bitmap to be created. The values of these are as given in gCREATE above (0 for 2 colour-mode, 1 for 4 colour-mode and 2 for 16 colour-mode). By default the graphics mode of a bitmap is 2-colour.

Note again that 64 drawables (including the default window) may be open at any time. Although as mentioned above, using lots of windows should be avoided in practice, you can sensibly use as many bitmaps as you need up to the maximum.

GGREY

Usage: Unchanged from OPL16

As stressed above, there is no longer any concept of a grey plane in OPL32, just different colours. Thus mode% of 1 just sets the foreground colour of the current drawable to light grey. mode% of any other value sets the foreground colour to black (the default). mode% of 2 (drawing to both grey and black planes) obviously has no meaning in OPL32.

GLINETO AND GLINEBY

Usage: `gLineTo x%,y%`

`gLineBy dx%,dy%`

In SIBO, the window server drew lines with the point at the bottom right omitted. OPL32 does not attempt to conform to this: it would be inefficient and would generally cause more problems than the incompatibility itself.

The Series 5 window server has the simple rule that it never draws the end point:

for `gLineTo x%,y%`, point (x%,y%) is not drawn

for `gLineBy dx%,dy%`, point (gX+dx%,gY+dy%) is not drawn

Note, however, that OPL specially plots the point when the start and end-point coincide, avoiding the need for a new gPlot keyword.

GLOADBIT AND GSAVEBIT

Usage: Unchanged from OPL16

gLOADBIT and gSAVEBIT no longer add a default filename extension to the input argument name if none is provided. In OPL16, filenames with no extension had default extension .PIC added.

Note that gLOADBIT loads EPOC Picture files, which are naturally in the same file format that is saved by gSAVEBIT. EPOC Picture files can also be generated using the PC tool BMCONV.EXE or by exporting files created by the Sketch application. These are called multi-bitmap files (MBMs), though often containing just one bitmap as in the case of gSAVEBIT or Sketch files, and are often given an extension .MBM. See ICON for detailed information on the use of BMCONV.

GLOADFONT AND GUNLOADFONT

Usage: `fileId%=gLOADFONT(file$)`

gLOADFONT no longer returns a font ID that can be used by gFONT. Instead it returns the file ID, which can be used only with gUNLOADFONT. The maximum number of font files which may be loaded at any one time is 16.

To use the fonts in a loaded font file you need to use their published UIDs, for example:

```
fileId%=gLOADFONT(“Music1”)
gFONT KMusic1Font1&
...
gUNLOADFONT fileId%
```

Attempting to unload a font which was not loaded now causes an ‘Invalid argument’ error.

GPEEKLINE

Usage: `gPEEKLINE id%,x%,y%,var arr%(),len%,mode%`

gPEEKLINE has an extra optional parameter mode% to specify the grey-scale mode which can take the following values:

-1 for black and white (black pixel sets bit),

0 for black and white (white pixel sets bit),

1 for 4-colour mode (white pixel sets bits),

2 for 16-colour mode (white pixel sets bits).

The default mode% is -1. For 4 and 16-colour modes, 2 and 4 bits per pixel respectively are used. This is to enable the colour of the pixel to be assertained from the bits which are set. White results in all 2 or 4 bits being set, while black sets none of them. For example, with the colour set by

```
gCOLOR 16,16,16
```

(see below for description of this function) a pixel of a line would peek as 0001 in binary. While a pixel of a line with the colour set to

```
gCOLOR 80,80,80
```

would result in the value 0101 in binary when peeked.

Note that this has the further implication that the array size allowed must be adjusted depending on the colour mode: it must be at least twice as long as the array needed for black and white if the line you wish to peek in 4-colour mode and four times as long in 16-colour mode. Note also that $8000 ORed with the id% no longer reads grey and if used will raise an ‘Invalid arguments’ error.

GXBORDER

Usage: `gXBORDER type%,flags%`

`gXBORDER type%,flags%,w%,h%`

This draws a border on the current drawable.

type%=2 for drawing on the Series 5.

Values for flags% are,

None	$00
Single Black	$01
Shallow Sunken	$42
Deep Sunken	$44
Deep Sunken With Outline	$54
Shallow Raised	$82

Deep Raised	$84
Deep Raised With Outline	$94
Vertical Bar	$22
Horizontal Bar	$2a

The constants for these values supplied in Const.oph are KBordSingleBlack% etc.

GRAPHICS - NEW KEYWORDS

GCOLOR

Usage: `gCOLOR red%,green%,blue%`

The red%,green%,blue% values specify a colour which will be mapped to white, black or one of the greys on non-colour screens. Note that if the values of red%, green% and blue% are equal, then a pure grey results, ranging from black (0) to white (255).

GINFO32

Usage: `gINFO32 var i&()`

This replaces gINFO var i&() because the information available has changed. i&() must now have 48 elements (although elements 37 to 48 are currently unused). The same information is returned to the array elements as for gINFO in OPL16 except for the following,

i&(1)	reserved
i&(2)	reserved
i&(9)	the font Uid as used in gFONT. The font name is not supplied.
i&(10-17)	unused
i&(30)	graphics colour-mode of current window (see gCREATE)
i&(31)	gCOLOR red% of foreground
i&(32)	gCOLOR green% of foreground
i&(33)	gCOLOR blue% of foreground
i&(34)	gCOLOR red% of background
i&(35)	gCOLOR green% of background
i&(36)	gCOLOR blue% of background.

GCIRCLE

Usage: `gCIRCLE radius%`

`gCIRCLE radius%,fill%`

This draws a circle with the centre at the current position in the current drawable. If the value of radius% is negative then no circle is drawn.

If the fill% argument is supplied and if fill%<>0 then the circle is filled.

GELLIPSE

Usage: `gELLIPSE hRadius%,vRadius%`

`gELLIPSE hRadius%,vRadius%,fill%`

This draws an ellipse with the centre at the current position in the current drawable. hRadius% is the horizontal distance in pixels from the centre of the ellipse to the left (and right) of the ellipse. vRadius% is the vertical distance from the centre of the ellipse to the top (and bottom). If the length of either radius is less than zero, then no ellipse is drawn.

If the fill% argument is supplied and if fill%<>0 then the ellipse is filled.

GSETPENWIDTH

Usage: `gSETPENWIDTH width%`

Sets the pen width in the current drawable to width% pixels.

OPX OVERVIEW

OPL32 uses language extensions provided in separate DLLs written specially for OPL support. These DLLs have the file extension OPX. A maximum of 255 OPXs can be used in any one OPL32 module and any OPX may have up to 65535 procedures defined in it. If these values are exceeded then errors are reported at translate-time (see the *New error codes* section in 'General Features of OPL32').

OPX HEADER FILES

The OPX supplier provides an OXH header file. This provides the declaration for the OPX, specifying its name UID and version number (see below), followed by its procedure prototypes.

The new INCLUDE keyword is used to include these header files. Alternatively, the declaration can be inserted directly into the OPL file, i.e.

```
DECLARE OPX <opxName>,<uid>,<version>
   <protoType1> : <ordinal1>
   <protoType2> : <ordinal2>
   ...
END DECLARE
```

where,

<name> is the name of the OPX without the .OPX extension. The OPX is stored in a \System\Opx\ folder on any drive, with the drives scanned from Y: to A: and then Z: if no path is specified. This allows ROM OPXs (in Z:) to be overridden if required.

<uid> is the UID of the OPX. The specification of a UID as well as a name guards against OPXs being with the same name being confused, which could otherwise cause serious problems. The UID is checked on loading the OPX and a 'Not supported' error will result if the UIDs in the header file and in the OPX itself do not match.

<version> is the version number of the OPX. The version number of an OPX will be increased when any new procedures are added. OPL will refuse to load an OPX which reports that it can't support the version number given in the declaration. The version number expressed in hex is e.g. $0100 to represent version 1.00, $0102 for version 1.02 etc. In general, an OPX supplier will increment the 2 low digits (the so-called minor version number) for backward compatible changes, and will increment the 2 high digits for major incompatible changes.

<prototype> specifies the name, return type and parameters of an OPX procedure in the same way as for an OPL procedure when using the EXTERNAL keyword. Numeric parameters to OPX procedure can be passed *by reference* using BYREF. This means that the value of the variable **can** be changed by the OPX procedure. The OPX procedure prototype is followed additionally by a colon and then the

<ordinal> specifies the ordinal number of the procedure in the OPX itself. This is used to call the correct procedure, i.e. the OPX is *linked by ordinal*.

If an OPX procedure name clashes with one of your OPL proecedures, or with that of another OPX procedure, you can make a copy of the OXH file and change the name of the offending procedures, but **not** the return type, parameter list or ordinal. You should then include this new OXH file in your module and the new name can then be used to refer to the procedure in your code.

For example, consider the OPX declaration,

```
DECLARE OPX XXOpx,XXOpxUid&,XXOpxVersion&
   XXClose%:(id&) : 1
   ProcWithLongParam:(aLong&) : 2
   AddOneToParams:(BYREF par1%,BYREF par2&, BYREF par3) : 3
END DECLARE
```

If a short integer is passed to ProcWithLongParam:, the translator will automatically convert it to a long integer.

The AddOneToParams: procedure adds one to each of the parameters. This is possible because the parameters are passed by reference. Parameters passed by reference must be variables rather than constant values because the OPX will write back to the variable. The variable type must always be the same as given in the declaration.

CALLBACKS FROM OPX PROCEDURES

An OPX procedure can call back to a module procedure. This is often useful if the OPX needs some information that is not known when the OPX procedure is initially called, or if it requires a large amount of data which needs to be sent piecemeal.

The OPX provider will specify the exact form of the procedure which you must provide.

OPXS RELEASED IN VERSION 1 OF THE SERIES 5

Procedures for handling the following are provided in the ROM:

- Date / time extras
- System - a variety pf procedures for system control, e.g. backlight control, sound control, file and application control, etc.
- Bitmaps - for use with buttons and Sprites.
- Sprites
- Database extras

These OPXs and their OXHs have default paths in the ROM (Z:), but the full path for any OPX may be supplied. These are \System\Opl\ for the header files and \System\Opx\ for the OPXs themselves. The OPX header files are stored in the ROM, but may be created in RAM by using the ‘Create standard file’ option in the ‘Tools’ menu in the Program editor.

With these OPXs, the OPL programmer is sometimes given direct access to objects via pointers (for efficiency). Otherwise sprites, for example, could not be set up using an array of IDs. These objects can be explicitly deleted to free memory or else they will be deleted when the program exits

DATE OPX

To use this OPX, you must included the header file Date.oxh, which contains the following declaration:

```
DECLARE OPX DATE,KUidOpxDate&,KOpxDateVersion%
   DTNewDateTime&:(year&,month&,day&,hour&,minute&,second&,micro&) : 1
   DTDeleteDateTime:(id&) : 2
   DTYear&:(id&) : 3
   DTMonth&:(id&) : 4
   DTDay&:(id&) : 5
   DTHour&:(id&) : 6
   DTMinute&:(id&) : 7
   DTSecond&:(id&) : 8

   DTMicro&:(id&) : 9
   DTSetYear:(id&,year&) : 10
   DTSetMonth:(id&,month&) : 11
   DTSetDay:(id&,day&) : 12
   DTSetHour:(id&,hour&) : 13
   DTSetMinute:(id&,minute&) : 14
   DTSetSecond:(id&,second&) : 15
   DTSetMicro:(id&,micro&) : 16
   DTNow&: : 17
   DTDateTimeDiff:(start&,end&,BYREF year&,BYREF month&,BYREF day&,BYREF
        hour&,BYREF minute&,BYREF second&,BYREF micro&) : 18
   DTYearsDiff&:(start&,end&) : 19
   DTMonthsDiff&:(start&,end&) : 20
   DTDaysDiff&:(start&,end&) : 21
   DTHoursDiff&:(start&,end&) : 22
   DTMinutesDiff&:(start&,end&) : 23
   DTSecsDiff&:(start&,end&) : 24
   DTMicrosDiff&:(start&,end&) : 25
   DTWeekNoInYear&:(id&,yearstart&,rule&) : 26
   DTDayNoInYear&:(id&,yearstart&) : 27
   DTDayNoInWeek&:(id&) : 28
   DTDaysInMonth&:(id&) : 29
   DTSetHomeTime:(id&) : 30
   LCCountryCode&: : 31
   LCDecimalSeparator$: : 32
   LCSetClockFormat:(format&) : 33
   LCClockFormat&: : 34
   LCStartOfWeek&: : 35
   LCThousandsSeparator$: : 36
END DECLARE
```

DTNEWDATETIME&:

Usage: `id&=NEWDATETIME&:(year&,month&,day&,hour&,minute&,second&,micro&)`

This creates a new date/time object which contains all the supplied date/time components and returns a handle id& for it.

The year is stored as the usual four figure year, eg. 1997.

The month is stored as 1 for January, 2 for Febrauary, etc.

The day is stored as the day number in the month.

The hour is the hour of the day in 12 or 24 hour clock according to the system setting.

The minutes, seconds and microseconds are stored as the usual values 0 to 59 for minutes and seconds and 0 to 999 for microseconds.

See DTDELETEDATETIME:.

DTDELETEDATETIME:

Usage: `DELETEDATETIME:(id&)`

This deletes the date/time object with handle id&. This should be called when a date/time object is no longer needed. Date/time objects will be deleted automatically on unloading the Date OPX or the module which uses it.

See DTNEWDATETIME&:, DTNOW&:.

DTYEAR&:

Usage: `y&=DTYEAR&:(id&)`

This returns the year component y& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTMONTH&:

Usage: `m&=DTMONTH&:(id&)`

This returns the month component m& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTDAY&:

Usage: `day&=DTDAY&:(id&)`

This returns the day component day& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTHOUR&:

Usage: `h&=DTHOUR&:(id&)`

This returns the hour component h& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:

DTMINUTE&:

Usage: `m&=DTMINUTE&:(id&)`

This returns the minutes component m& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTSECOND&:

Usage: `s&=DTSECOND&:(id&)`

This returns the seconds component s& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTMICRO&:

Usage: `m&=DTMICRO&:(id&)`

This returns the microseconds component m& which is stored in the date/time object with handle id&.

See DTNEWDATETIME&:.

DTSETYEAR:

Usage: `DTSETYEAR:(y&,id&)`

This sets the year component which is stored in the date/time object with handle id& to y&.

See DTNEWDATETIME&:.

DTSETMONTH:

Usage: `DTSETMONTH:(m&,id&)`

This sets the month component which is stored in the date/time object with handle id& to m&.

See DTNEWDATETIME&:.

DTSETDAY:

Usage: `DTSETDAY:(day&,id&)`

This sets the day component which is stored in the date/time object with handle id& to day&.

See DTNEWDATETIME&:.

DTSETHOUR:

Usage: `DTSETHOUR:(h&id&)`

This sets the hour component which is stored in the date/time object with handle id& to h&.

See DTNEWDATETIME&:.

DTSETMINUTE:

Usage: `DTSETMINUTE:(m&,id&)`

This sets the minutes component which is stored in the date/time object with handle id& to m&.

See DTNEWDATETIME&:.

DTSETSECOND:

Usage: `DTSETSECOND:(s&,id&)`

This sets the seconds component which is stored in the date/time object with handle id& to s&.

See DTNEWDATETIME&:.

DTSETMICRO:

Usage: `DTSETMICRO:(m&,id&)`

This sets the microseconds component which is stored in the date/time object with handle id& to m&.

See DTNEWDATETIME&:.

DTNOW&:

Usage: `id&=DTNOW&:`

This creates a new date/time object which contains all the date/time components of the current time and returns a handle id& for it.

Example: Timing a loop

```
...
start&=DTNow&:
WHILE condition
...
ENDWH
end&=DTNow&:
PRINT “Time to do loop was”,DTMicrosDiff&:(start&,end&)
...
```

See DTNEWDATETIME&:.

DTDATETIMEDIFF:

Usage: `DTDATETIMEDIFF:(start&,end&,BYREF year&,BYREF month&,BYREF day&,BYREF hour&,BYREF minute&,BYREF second&,BYREF micro&)`

This calculates the exact difference between two date/time objects with handles start& and end& in the form of a date/time object. The difference is returned in the variables year&, month& etc.

DTYEARSDIFF&:

Usage: `diff&=DTYEARSDIFF&:(start&, end&)`

This returns the difference diff& in whole years between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTMONTHSDIFF&:

Usage: `diff&=DTMONTHSDIFF&:(start&, end&)`

This returns the difference diff& in whole months between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTDAYSDIFF&:

Usage: `diff&=DTDAYSDIFF&:(start&, end&)`

This returns the difference diff& in whole days between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTHOURSDIFF&:

Usage: `diff&=DTHOURSDIFF&:(start&, end&)`

This returns the difference diff& in whole hours between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTMINUTESDIFF&:

Usage: `diff&=DTMINUTESDIFF&:(start&, end&)`

This returns the difference diff& in whole minutes between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTSECONDSDIFF&:

Usage: `diff&=DTSECONDSDIFF&:(start&, end&)`

This returns the difference diff& in whole seconds between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:.

DTMICROSDIFF&:

Usage: `diff&=DTMICROSDIFF&:(start&, end&)`

This returns the difference diff& in whole microseconds between the two date/time objects with handles start& and end&.

See DTNEWDATETIME&:, DTNOW&:.

DTWEEKNOINYEAR&:

Usage: `w&=DTWEEKNOINYEAR&:(id&,yearstart&,rule&)`

This returns the week number in the year of the date/time object with handle id&. The first day of the year is specified by the date/time object with handle yearstart&. This would usually be set to 1 January in the appropriate year, but also allows you to set the star of the year to the beginning of the financial year or the academic year, for example.

rule& can take three values (0,1,2), allowing the week number to be calculated by one of three different rules:

0 means the first day of the year is always in week one,

1 requires the first week of the year to have at least four days in it,

2 requires the first week of the year to have the full seven days.

The Agenda application and other Series 5 applications use the rule with value 1 by default.

DTDAYNOINYEAR&:

Usage: `d&=DTDAYNOINYEAR&:(id&,yearstart&)`

This returns the day number in the year of the date/time object with handle id&. The first day of the year is specified by the date/time object with handle yearstart&.

DTDAYNOINWEEK&:

Usage: `n&=DTDAYNOINWEEK&:(id&)`

This returns the day number in the week (1-7) of the date/time object with handle id&.

DTDAYSINMONTH&:

Usage: `n&=DTDAYSINMONTH&:(id&)`

This returns the number of days in the month of the month specified by the month component of the date/time object with handle id&.

DTSETHOMETIME:

Usage: `DTSETHOMETIME:(id&)`

This sets the system time to the time specified in the date/time object with handle id&.

LCCOUNTRYCODE&:

Usage: `cc&=LCCOUNTRYCODE&:`

This returns the country code for the current system home country (LC stands for 'Locale'), which may be used to select country-specific data. The country code for any given country is the international dialling prefix for that country.

LCDECIMALSEPARATOR$:

Usage: `decSep$=LCDECIMALSEPARATOR$:`

This returns the decimal separator (the character used in decimal numbers to separate whole part from fractional part) according to the local system setting.

LCSETCLOCKFORMAT:

Usage: `LCSETCLOCKFORMAT:(format&)`

This sets the system clock format to either digital or analog. If format&=0 then the clock is set to analog or if it is 1 then the clock is set to digital.

LCCLOCKFORMAT&:

Usage: `format&=LCCLOCKFORMAT&:`

This returns the current system clock format. The procedure returns 0 if the system clock is analog and 1 if it is digital.

LCSTARTOFWEEK&:

Usage: `start&=LCSTARTOFWEEK&:`

This returns the day of the week which has been set as the first day of the week in the system setting. A return value of 1 indicates Monday, 2 Tuesday, and so on.

LCTHOUSANDSSEPARATOR$:

Usage: `thouSep$=LCTHOUSANDSSEPARATOR$:`

This returns the thousands separator (the character used to separate every three digits of a large number) according to the local system setting.

SYSTEM OPX

To use this OPX, you must included the header file System.oxh, which contains the following declaration:

```
DECLARE OPX SYSTEM,KUidOpxSystem&,KOpxSystemVersion%
  BackLightOn&: : 1
  SetBackLightOn:(state&) : 2
  SetBackLightOnTime:(seconds&) : 3
  SetBacklightBehavior:(behaviour&) : 4
  IsBacklightPresent&: : 5
  SetAutoSwitchOffBehavior:(behaviour&) : 6
  SetAutoSwitchOffTime:(seconds&) : 7
  SetActive:(state&) : 8
  ResetAutoSwitchOffTimer: : 9
  SwitchOff: : 10
  SetSoundEnabled:(state&) : 11
  SetSoundDriverEnabled:(state&) : 12
  SetKeyClickEnabled:(state&) : 13
  SetPointerClickEnabled:(state&) : 14
  SetDisplayContrast:(value&) : 15
  MaxDisplayContrast&: : 16
  IsReadOnly&:(file$) : 17
  IsHidden&:(file$) : 18
  IsSystem&:(file$) : 19
  SetReadOnly:(file$,state&) : 20
  SetHiddenFile:(file$,state&) : 21
  SetSystemFile:(file$,state&) : 22
  VolumeSize&:(drive&) : 23
  VolumeSpaceFree&:(drive&) : 24
  VolumeUniqueID&:(drive&) : 25
  MediaType&:(drive&) : 26
  GetFileTime:(file$,DateTimeId&) : 27
  SetFileTime:(file$,DateTimeId&) : 28
  DisplayTaskList: : 29
  SetComputeMode:(State&) : 30
  RunApp&:(lib$,doc$,tail$,cmd&) : 31
  RunExe&:(name$) : 32
  LogonToThread:(threadId&,BYREF statusWord&) : 33
  TerminateCurrentProcess:(reason&) : 34
```

```
  TerminateProcess:(proc$,reason&) : 35
  KillCurrentProcess:(reason&) : 36
  KillProcess:(proc$,reason&) : 37
  PlaySound:(file$,volume&) : 38
  PlaySoundA:(file$,volume&,BYREF statusWord&) : 39
  StopSound&: : 40
  Mod&:(left&,right&) : 41
  XOR&:(left&,right&) : 42
  LoadRsc&:(file$) : 43
  UnLoadRsc:(id&) : 44
  ReadRsc$:(id&) : 45
  ReadRscLong&:(id&) : 46
  CheckUid$:(Uid1&,Uid2&,Uid3&) : 47
  SetPointerGrabOn:(WinId&,state&) : 48
  MachineName$: : 49
  MachineUniqueId:(BYREF high&,BYREF low&) : 50
  EndTask&:(threadId&,previous&)  : 51
  KillTask&:(threadId&,previous&) : 52
  GetThreadIdFromOpenDoc&:(doc$,BYREF previous&) : 53
  GetThreadIdFromAppUid&:(uid&,BYREF previous&) : 54
  SetForeground: : 55
  SetBackground: : 56
  SetForegroundByThread&:(threadId&,previous&) : 57
  SetBackgroundByThread&:(threadId&,previous&) : 58
  GetNextWindowGroupName$:(threadId&,BYREF previous&) : 59
  GetNextWindowId&:(threadId&,previous&) : 60
  SendKeyEventToApp&:(threadId&,previous&,code&,scanCode&,modifiers&,repeats&)
          : 61
  IrDAConnectToSend&:(protocol$,port&) : 62
  IrDAConnectToReceive:(protocol$,port&,BYREF statusW&) : 63
  IrDAWrite:(chunk$,BYREF statusW&) : 64
  IrDARead$: : 65
  IrDAReadA:(stringAddr&,BYREF statusW&): 66
  IrDAWaitForDisconnect: : 67
  IrDADisconnect: : 68
  MainBatteryStatus&: :69
  BackupBatteryStatus&: :70
  CaptureKey&:(keyCode&,mask&,modifier&) :71
  CancelCaptureKey:(handle&) :72
  SetPointerCapture:(winId&, flags&) :73
  ClaimPointerGrab:(winId&, state&) :74
  OpenFileDialog$:(SeedFile$,Uid1&,Uid2&,Uid3&) : 75
  CreateFileDialog$:(SeedPath$) : 76
  SaveAsFileDialog$:(SeedPath$,BYREF UseNewFile%) : 77
END DECLARE
```

BACKLIGHTON&:

Usage: `backLight&=BACKLIGHTON&:`

This procedures returns -1 if the backlight is switched on or 0 if it is switched off.

SETBACKLIGHTON:

Usage: `SETBACKLIGHTON:(state&)`

This procedure switches the backlight on if state&=1 or off if state&=0.

SETBACKLIGHTONTIME:

Usage: `SETBACKLIGHTONTIME:(seconds&)`

This sets the time in seconds (seconds&) that the backlight should remain on after it has been switched on.

SETBACKLIGHTBEHAVIOR:

Usage: `SETBACKLIGHTBEHAVIOR:(behavior&)`

This procedure sets the backlight's turning off behavior.

behavior&=0 sets the turning off of the backlight to be on a timer,

behavior&=1 sets the turning off of the backlight not to be on a timer.

ISBACKLIGHTPRESENT&:

Usage: `ret&=ISBACKLIGHTPRESENT&:`

This procedure returns -1 if there is a backlight present or 0 if there is not.

SETAUTOSWITCHOFFBEHAVIOR:

Usage: `SETAUTOSWITCHOFFBEHAVIOR:(behavior&)`

This procedure sets the machine's auto switch off behavior.

behavior&= 0 disables the machine's auto switch off mechanism,

behavior&= 1 sets the machine auto switch off to occur only when its batteries are low,

behavior&= 2 sets the machines auto switch off to occur always.

SETAUTOSWITCHOFFTIME:

Usage: `SETAUTOSWITCHOFFTIME:(seconds&)`

This sets the time in seconds (seconds&) for which the machine may remain switched on when it is not being used.

SETACTIVE:

Usage: `SETACTIVE:(state&)`

This sets the current process active if state&=1 or not active if state&=0. This will determine whether or not the machine can automatically turn off when the user is not using the machine: if the process is active then the machine will not automatically turn off.

RESETAUTOSWITCHOFFTIMER:

Usage:`RESETAUTOSWITCHOFFTIMER:(seconds&)`

This 'tickles' the machine's auto switch off timer, restarting its count down to switching off.

SWITCHOFF:

Usage:`SWITCHOFF:`

This switches off the machine.

As with the keyword OFF, you should be careful not to use SWITCHOFF: in a loop as it may then be impossible to switch the Series 5 back on without resetting it.

SETSOUNDENABLED:

Usage:`SETSOUNDENABLED:(state&)`

This enables the machine's sound if state&=1 or disables it if state&=0.

SETSOUNDDRIVERENABLED:

Usage:`SETSOUNDDRIVERENABLED:(state&)`

This switches the machines sound driver on if state&=1 or off if state&=0.

SETKEYCLICKENABLED:

Usage:`SETKEYCLICKENABLED:(state&)`

This determines whether or not a keypress makes a click. state&=1 enables the click and state&=0 disables it.

SETPOINTERCLICKENABLED:

Usage:`SETPOINTERCLICKENABLED:(state&)`

This determines whether or not a pointer event makes a click. (A pointer event occurs whenever the machine's screen is pressed.) state&=1 enables the click and state&=0 disables it.

SETDISPLAYCONTRAST:

Usage:`SETDISPLAYCONTRAST:(value&)`

This sets the contrast on the machine's screen. value& specifies the contrast value, which can be between zero and the maximum display contrast.

See MAXDISPLAYCONTRAST&:.

MAXDISPLAYCONTRAST&:

Usage:`maxContrast&=MAXDISPLAYCONTRAST&:`

This returns the maximum value to which the machine's display contrast can be set.

See SETDISPLAYCONTRAST:.

ISREADONLY&:

Usage: `readOnly&=ISREADONLY&:(file$)`

This procedure returns -1 if the file, file$, is a read-only file and 0 if it is not.

ISHIDDEN&:

Usage: `hidden&=ISHIDDEN&:(file$)`

This procedure returns -1 if the file, file$, is hidden by the system and 0 if it is not.

ISSYTEM&:

Usage: `system&=ISSYTEM&:(file$)`

This procedure returns -1 if the file, file$, is a system file and 0 if it is not.

SETREADONLY:

Usage: `SETREADONLY:(file$, state&)`

This sets the file, file$, to be read only if state&=1 or not read only if state&=0.

SETHIDDENFILE:

Usage: `SETHIDDENFILE:(file$, state&)`

This sets the file, file$, to be hidden by the system if state&=1 or not to be hidden if state&=0.

SETSYSTEMFILE:

Usage: `SETSYSTEMFILE:(file$, state&)`

Sets the file, file$, to be a system file if state&=1 or not to be a system file if state&=0.

VOLUMESIZE&:

Usage: `size&=VOLUMESIZE&:(drive&)`

This procedure returns the size in bytes of the storage space on the drive specified by drive&. drive& can take values 0 to 25. 0 specifies A:, 1 specifies B:, 2 specifies C: and so on.

VOLUMESPACEFREE&:

Usage: `free&=VOLUMESPACEFREE&:(drive&)`

This procedure returns the size in bytes of the storage space that is available for use on the drive specified by drive&. drive& can take values 0 to 25. 0 specifies A:, 1 specifies B:, 2 specifies C: and so on.

VOLUMEUNIQUEID&:

Usage: `volUid&=VOLUMEUNIQUEID&:(drive&)`

This procedure returns the unique identification number of the media on the drive specified by drive&. drive& can take values 0 to 25. 0 specifies A:, 1 specifies B:, 2 specifies C: and so on.

See MEDIATYPE&:.

MEDIATYPE&:

Usage: `media&=MEDIATYPE&:(drive&)`

This procedure returns the media type present on the drive specified by drive&. drive& can take values 0 to 25. 0 specifies A:, 1 specifies B:, 2 specifies C: and so on. The values that can be returned are defined as constants in System.oxh as follows:

```
CONST KMediaNotPresent& =0
CONST KMediaUnknown&    =1
CONST KMediaFloppy&          =2
CONST KMediaHardDisk&   =3
CONST KMediaCdRom&      =4
CONST KMediaRam&        =5
CONST KMediaFlash&      =6
CONST KMediaRom&        =7
CONST KMediaRemote&          =8
```

GETFILETIME:

Usage: `GETFILETIME:(file$,dateTimeId&)`

This procedure returns the time that the file, file$, was last modified into dateTimeId&. It is necessary to pass this procedure the ID of a date/time object which the procedure will then set to the required time. To obtain and read a date/time object see the 'Date OPX' section.

SETFILETIME:

Usage: `SETFILETIME:(file$,dateTimeId&)`

This sets the time that the file, file$, was last modified to dateTimeId&. It is necessary to pass this procedure the ID of a dateTime object which the procedure will use to set the time. To get a date/time object see the procedure DTNEWDATETIME&:, etc in the 'Date OPX' section.

DISPLAYTASKLIST:

Usage: `DISPLAYTASKLIST:`

This displays the system-wide task list. The user may then close a file, go to another file or close the dialog.

SETCOMPUTEMODE:

Usage: `SETCOMPUTEMODE:(state&)`

Changes the priority control for the current program.

The following constants are defined for state& in System.oxh:

```
CONST KComputeModeDisabled&  =0
CONST KComputeModeOn&        =1
CONST KComputeModeOff&       =2
```

This puts the current process into compute mode (state&=KComputeModeOn&) or takes it out of compute mode (state&=KComputeModeOff&). In compute mode, a process runs at the lower background priority even when it is the foreground process. Disabling compute mode control (state&=KComputeModeDisabled&) prevents the window server from changing the program's priority when it moves between background and foreground.

OPL runs in compute mode by default (as did OPL16) to support simple OPOs which cannot always be assumed to be well-behaved programs.

This default behaviour is necessary because OPL programs would not otherwise give any background programs a chance to run, simply by running in a tight loop. If your program doesn't run in a tight loop, i.e. if it calls GETEVENT32, GET, etc. regularly, you can use SETCOMPUTEMODE:(KComputeModeOff&).

Note that TBarInit: in Toolbar.opo sets compute mode off, since any program that has a toolbar shouldn't be running in a tight loop at any time.

RUNAPP&:

Usage: `thread&=RUNAPP&:(lib$,doc$,tail$,cmd&)`

This runs an application and returns a thread ID for the application. The thread ID can be used to logon to the application, to find out when and why it finished. This ID can also be used to locate the window group, end the task etc.

lib$ is the C++ application name.

doc$ is the document name, if any, to pass to the application.

tail$ is the tail end, needed only by certain applications such as OPL.

cmd& can take values:

0 = Open

1= Create

2 = Run

3 = Background

The values: 0,1 and 2 are as for CMD$(3) (see the 'General Changed Keywords' section). The value 3 will run the application in the background.

Example: for running an OPL program:

```
k&=RUNAPP&:( "OPL", "", "RZ:\System\Opl\Toolbar.opo",2)
```

Note that tail$ contains a leading R: this is required by OPL.

See LOGONTOTHREAD:.

RUNEXE&:

Usage: `RUNEXE&:(file$)`

This runs an executable EXE file file$ and returns its thread ID. The thread ID can then be used to logon to the thread and find out when and why it finished.

See LOGONTOTHREAD:.

LOGONTOTHREAD:

Usage: `LOGONTOTHREAD:(threadId&, statusWord&)`

This logs on to the thread with ID threadId& and sets the status word statusWord& when the thread has completed. As for other 32-bit status words, statusW& will be set to &80000001 for pending and 0 for successful completion.

TERMINATECURRENTPROCESS:

Usage:`TERMINATEPROCESS:(reason&)`

This terminates the current process giving it the reason reason&. This causes the process to receive a shutdown message. reason& may take any value, with zero used to mean no errors.

TERMINATEPROCESS:

Usage:`TERMINATEPROCESS:(proc$,reason&)`

This terminates the process proc$ giving it the reason reason&. This causes the process to receive a shutdown message. reason& may take any value, with zero used to mean no errors.

KILLCURRENTPROCESS:

Usage:`KILLCURRENTPROCESS:(reason&)`

This kills the current process giving it the reason reason&. The process is killed *without* receiving a shutdown message. reason& may take any value, with zero used to mean no errors.

KILLPROCESS:

Usage:`KILLPROCESS:(proc$,reason&)`

This kills the process proc$ giving it the reason reason&. The process is killed *without* receiving a shutdown message. reason& may take any value, with zero used to mean no errors.

PLAYSOUND:

Usage:`PLAYSOUND:(file$,volume&)`

This plays the file file$ which may be a sound file or an alarm file. Does not return until the sound has completed.

volume& specifies the volume at which the file should be played, 0 specifies no volume and 3 specifies maximum volume. The play will be synchronous (i.e. the OPL program will stop running until the file has finished playing). For asynchronous sound playing see PLAYSOUNDA:.

PLAYSOUNDA:

Usage:`PLAYSOUNDA:(file$,volume&,statusWord&)`

This plays the file file$, which may be a sound file or an alarm file, asynchronously. It returns immediately, with the status word being set on completion or cancellation of the sound.

volume& specifies the volume at which the file should be played, 0 specifies no volume and 3 specifies maximum volume. The play will be asynchronous (i.e. the OPL program will continue running while the sound file is playing). statusWord& is the variable which will contain information about the state of the sounds file play. For synchronous sound playing see PLAYSOUND:.

STOPSOUND&:

Usage:`ret&=STOPSOUND&:`

This stops the sound file that is currently playing. The return value is 1 if there was a sound file playing and 0 if not.

See PLAYSOUND:, PLAYSOUNDA:.

MOD&:

Usage: `rem&=MOD&:(left&,right&)`

This returns left& modulo right& (i.e.the remainder of left& divided by right&).

XOR&:

Usage: `res&=XOR&:(left&,right&)`

This procedure returns the exclusive OR of left& and right&. A bit in the result has value 1 if the corresponding bit is set in *either* left& or in right&, but *not* in both, otherwise it is 0.

E.g. with left&=5 (binary 00000101)and right&=3 (binary 00000011), XOR&:(left&,right&) returns 6 (binary 00000110).

left&	00000101
right&	00000011
`XOR&:(left&,right&)`	00000110

This is often used to invert particular bits in an integer. So left&=XOR&:(left&,3) inverts bits 0 and 1 in left&, where bit 0 is the rightmost bit, and leaves the other bits alone.

LOADRSC&:

Usage: `id&=LOADRSC&:(file$)`

This loads the resource file file$ and returns a handle for it.

See UNLOADRSC:, READRSC$:.

UNLOADRSC:

Usage: `UNLOADRSC:(id&)`

This unloads the resource file whose handle is id&.

See LOADRSC&:.

READRSC$:

Usage: `string$=READRSC$:(id&)`

This reads the resource in a resource file specified by id& which must already be loaded.

See LOADRSC&:.

READRSCLONG&:

Usage: `long&=READRSCLONG&:(id&)`

This procedure reads a 32-bit value from a resource file specified by the id& which must already be loaded.

See LOADRSC&:.

CHECKUID$:

Usage: `CHECKUID$:(Uid1&,Uid2&,Uid3&)`

This returns a string which is a unique product of the three supplied UIDs.

SETPOINTERGRABON:

Usage: `SETPOINTERGRABON:(WinId&,state&)`

You can use this procedure to enable (or disable) pointer grabs in a window. After this function has been called with state&=1, any down event in this window will cause a pointer grab, terminated by the next corresponding up event. The terminating up event is also sent to the window with the grab.

This function would typically be used for drag-and-drop events, and would typically be called after window creation so that pointer grab is enabled for the lifetime of the window. state&=0 disables the grab.

See CLAIMPOINTERGRAB:.

MACHINENAME$:

Usage: `name$=MACHINENAME$:`

This procedure returns the machine name.

MACHINEUNIQUEID:

Usage: `MACHINEUNIQUEID:(BYREF high&,BYREF low&)`

This procedure sets high& and low& (which are passed BYREF) with high and low parts of the machine's unique ID respectively.

ENDTASK&:

Usage: `ret&=ENDTASK&:(threadId&,previous&)`

This procedure sends a *shutdown* message to a window group in the thread, `threadId&`.

`previous&` specifies the window group in the thread that is previous to the one required. Hence, setting `previous&` to 0 will specify the first window group in the thread.

The value returned by this procedure is the value of the window group on which action was taken, or -1 if a window group following that specified by `previous&` was not present. This return value could be passed back to this procedure to end the next window group in the thread.

An error will be raised if the window group is busy, does not respond to such requests or is in a system thread.

See also KILLTASK&:.

KILLTASK&:

Usage: `ret&=KILLTASK&:(threadId&,previous&)`

This procedure will shutdown the window group that follows the window group specified by previous& in the thread, threadId&. It will ignore any wishes of the window group not to be shutdown.

The value returned by this procedure is the value of the window group on which action was taken, or -1 if a window group following that specified by previous& was not present. This return value could be passed back to this procedure to end the next window group in the thread.

See also ENDTASK&:.

GETTHREADIDFROMOPENDOC&:

Usage: `threadId&=GETTHREADIDFROMOPENDOC&:(doc$,BYREF previous&)`

This procedure returns the thread ID threadId& of the thread that contains the open document doc$.

doc$ should contain the full path of the open document and could for example be “c:\opl\prog.opo”.

previous&, passed by reference, is useful for situations where the document may be open more than once. Setting previous& to zero will cause this procedure to find the first window group that contains doc$. If one is found, previous& will be set to the window group value of that found document. This value could then passed back to this procedure as previous&, and the next window group instance of doc$ can be found, and so on. If this procedure sets previous& to -1 then doc$ could not be found after previous&. An error will also be raised if the open document is not found.

GETTHREADIDFROMAPPUID&:

Usage: `threadId&=GETTHREADIDFROMAPPUID&:(uid&,BYREF previous&)`

This procedure returns the thread ID threadId& of the thread that contains a running instance of the application with UID uid&.

previous&, passed by reference, is useful for situations where the application may be running more than once. Setting previous& to zero will cause this procedure to find the first window group that is an instance of this application. If one is found, previous& will be set to the value of that window group. This value could then passed back to this procedure as previous&, and the instance of the application can be found, and so on. If no instance of this application following previous& can be found, then previous& will be set to -1. An error will also be raised if a running instance of the application is not found.

SETFOREGROUND:

Usage: `SETFOREGROUND:`

This moves the calling process to foreground.

See SETBACKGROUND:.

SETBACKGROUND:

Usage: `SETBACKGROUND:`

This moves the calling process to background.

See SETFOREGROUND:.

SETFOREGROUNDBYTHREAD&:

Usage: `ret&=SETFOREGROUNDBYTHREAD&:(threadId&,previous&)`

This moves the window group after previous& in the thread threadId& to foreground. Using previous&=0 specifies the first window group in the thread.

The value returned will be the window group on which action was taken, or -1 if the following window group was not found. This return value could be passed back to this procedure as a value for previous& to put the next window group in threadId& to foreground.

See SETBACKGROUNDBYTHREAD&:

SETBACKGROUNDBYTHREAD&:

Usage: `ret&=SETBACKGROUNDBYTHREAD&:(threadId&,previous&)`

This moves the window group after previous& in the thread threadId& to background. Using previous&=0 specifies the first window group in the thread.

The value returned will be the window group on which action was taken, or -1 if the following window group was not found. This return value could be passed back to this procedure as a value for previous& to put the next window group in threadId& to background.

See SETFOREGROUNDBYTHREAD&:.

GETNEXTWINDOWGROUPNAME$:

Usage: `name$=GETNEXTWINDOWGROUPNAME$:(threadId&,BYREF previous&)`

This procedure returns the name of the window group that follows the window group previous& in the thread threadId&. If previous& is 0 the procedure will return the name of the first window group found.

See GETNEXTWINDOWID&:.

GETNEXTWINDOWID&:

Usage: `winId&=GETNEXTWINDOWID&:(threadId&,BYREF previous&)`

This procedure returns the ID of the window group that follows the window group previous& in the thread threadId&. If previous& is 0 the procedure will return the ID of the first window group found.

See GETNEXTWINDOWGROUPNAME$:.

SENDKEYEVENTTOAPP&:

Usage: `winId&=SENDKEYEVENTTOAPP&:(threadId&,previous&,code&,scanCode&,modifiers&,repeats&)`

This procedure sends a key event to the window group (application) that follows the window group with ID previous& (or the first window group if previous&=0), in the thread with ID threadId&. The key event is specified by code&, scanCode&, modifiers& and repeats&.

The value returned is the window group to which the key was sent.

IRDACONNECTTOSEND&:

Usage: `err&=IRDACONNECTTOSEND&:(protocol$,port&)`

This synchronous procedure sends out an infrared (IR) beam which looks for another infrared device. If another infrared device is looking to receive a beam then the devices will connect, otherwise this procedure will raise an error after a couple of seconds.

protocol$ can take either of the constants defined in System.oxh. These are KIrTinyTP$, used for communicating with other EPOC32 devices and KIrmux$ which is used for IR printing.

The value 8 is recommended for port& when communicating with other EPOC32 devices. Both devices will need to be using the same value for port&. port& must be 2 for connecting to IR printers.

If connecting to another EPOC32 device with this procedure the other device must connect by using IRDACONNECTTORECEIVE&:.

After connecting to another EPOC32 device you can both send and receive information.

After connecting to an IR printer use IRDAWRITE: to send text. When IRDADIASCONNECT: is called the printer will begin printing.

See IRDACONNECTTORECEIVE:, IRDAWRITE:, IRDAREAD:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDACONNECTTORECEIVE:

Usage: `IRDACONNECTTORECEIVE:(protocol$,port&,BYREF statusW&)`

This asynchronous procedure waits, without timing-out, for another EPOC32 device to attempt to connect via infrared (IR). This procedure takes the status word statusW&. When connetion occurs the status word is set to zero.

Both EPOC32 devices must be using the same port which is specified by port&. The recommended port is 8.

One of the devices must connect with this procedure and the other with IRDACONNECTTOSEND&:.

protocol$ specifies the IR protocol and should take the constant KIrTinyTP$, defined in System.oxh, when communicating with other EPOC32 devices.

After connecting to another EPOC32 device you can both send and receive information.

See IRDAWRITE:, IRDAREAD:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDAWRITE:

Usage: `IRDAWRITE:(chunk$,BYREF statusW&)`

This procedure sends a string, `chunk$`, via infrared to another device after connection has been made. The procedure is asynchronous and the success of sending the string is reported in `statusW&`.

See IRDACONNECTTOSEND&:, IRDACONNECTTORECEIVE:, IRDAREAD:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDAREAD$:

Usage: `chunk$=IRDAREAD$:`

This procedure is reads a text string via infrared from another device after connection has been made. The procedure is synchronous.

See IRDACONNECTTOSEND&:, IRDACONNECTTORECEIVE:, IRDWRITE:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDAREADA:

Usage: `IRDAREADA:(stringAddr&,BYREF statusW&)`

This procedure reads a text string via infrared from another device after connection has been made. The procedure is asynchronous with status word statusW&. The string passed by address should have a maximum length of 255 bytes. To pass a string by address use ADDR(string$).

See IRDACONNECTTOSEND&:, IRDACONNECTTORECEIVE:, IRDWRITE:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDAWAITFORDISCONNECT:

Usage: `IRDAWAITFORDISCONNECT:`

This procedure verifies, when disconnecting from another IR device, that the other device has received everything sent. It should therefore be called by the device that last sends some information.

This procedure should not be used with printers: use IRDADISCONNECT: instead.

See IRDACONNECTTOSEND&:, IRDACONNECTTORECEIVE:, IRDADISCONNECT:, IRDAWAITFORDISCONNECT:.

IRDADISCONNECT:

Usage: `IRDADISCONNECT:`

This procedure disconnects from another IR device. It should be called by the device that is the last to receive some information. IRDAWAITFORDISCONNECT: is used by the last device to send some information, except when printing via IR when IRDADISCONNECT: should be used to disconnect from the printer and commence printing.

See IRDACONNECTTOSEND&:, IRDACONNECTTORECEIVE:, IRDADISCONNECT: and IRDAWAITFORDISCONNECT:.

MAINBATTERYSTATUS&:

Usage: `stat&=MAINBATTERYSTATUS&:`

This procedure returns the current power of the main batteries. The return values, defined in System.oxh, as follows:

```
CONST KBatteryZero&     =0
CONST KBatteryVeryLow&  =1
CONST KBatteryLow&      =2
CONST KBatteryGood&     =3
```

BACKUPBATTERYSTATUS&:

Usage: `stat&=BACKUPMAINBATTERYSTATUS&:`

This procedure returns the current power of the backup batteries. The return values, defined in System.oxh, as follows:

```
CONST KBatteryZero&     =0
CONST KBatteryVeryLow&  =1
CONST KBatteryLow&      =2
CONST KBatteryGood&     =3
```

CAPTUREKEY&:

Usage:`handle&=CAPTUREKEY&:(keycode&,mask&,modifier&)`

This procedure allows the program that calls it to capture keys when it is not in foreground. The keys to be captured are specified using keycode&, mask& and modifier&. The value returned is a handle which can be passed to CANCELCAPTUREKEY: to cancel the capture.

The following constants are supplied in System.oxh to be used in conjuction with this procedure:

```
CONST KModifierAutorepeatable&=&00000001
CONST KModifierKeypad&=&00000002
CONST KModifierLeftAlt&=&00000004
```

```
CONST KModifierRightAlt&=&00000008
CONST KModifierAlt&=&00000010
CONST KModifierLeftCtrl&=&00000020
CONST KModifierRightCtrl&=&00000040
CONST KModifierCtrl&=&00000080
CONST KModifierLeftShift&=&00000100
CONST KModifierRightShift&=&00000200
CONST KModifierShift&=&00000400
CONST KModifierLeftFunc&=&00000800
CONST KModifierRightFunc&=&00001000
CONST KModifierFunc&=&00002000
CONST KModifierCapsLock&=&00004000
CONST KModifierNumLock&=&00008000
CONST KModifierScrollLock&=&00010000
CONST KModifierKeyUp&=&00020000
CONST KModifierSpecial&=&00040000
CONST KModifierDoubleClick&=&00080000
CONST KModifierPureKeycode&=&00100000
CONST KAllModifiers&=&001fffff
```

See CANCELCAPTUREKEY&:.

CANCELCAPTUREKEY:

Usage: `CANCELCAPTUREKEY(handle&)`

This procedure cancels an outstanding key capture which is specified by the handle handle&.

See CAPTUREKEY&:.

SETPOINTERCAPTURE:

Usage: `SETPOINTERCAPTURE:(winId&,flags&)`

This sets the (OPL) window with window ID winId& to capture pointer events.

flags& can take a summed combination of the following

Capture Disabled	0
Capture Enabled	&01
Capture Not Drag Drop	&02

When a window captures pointer events, the position of any events received will be relative to its origin, i.e. the top left corner of that window, i.e. the window's 0,0 coordinate becomes the origin for pointer events. Events don't need to be inside the window to be registered. For example, if the screen was touched 1 pixel above and 1 pixel to the left of a window that was capturing pointer events, the pointer event would be returned with the position of the event being -1,-1 .

'Capture Not Drag Drop' will cause events of a drag-and-drop nature to be sent to the window that would have received them had the pointer not been captured..

CLAIMPOINTERGRAB:

Usage: `CLAIMPOINTERGRAB:(winId&,state&)`

If a pointer grab is already in effect in another window, a window, winId&, can claim the pointer grab from that window by calling this function. All subsequent events will be delivered to this window, instead of the window that originally had the pointer grab. The next up event terminates the pointer grab, and is also delivered to the window that claimed the grab, if state& is set to 1.

This function would typically be used where clicking in a window pops up another window, and where the popped-up window wishes to grab the pointer as though the original click had been in that window.

To summarise the values of state& for the OPL window, winId&

Don't deliver the following up event to this window = 0

Deliver the following up event to this window = 1

See SETPOINTERGRABON:

OPENFILEDIALOG$:

Usage: `file$=OPENFILEDIALOG$:(seedFile$,uid1&,uid2&,uid3&)`

This procedure displays the standard 'Open file' dialog. seedFile$ provides a starting file which is displayed when the dialog is opened. So, for example, if seedFile$="C:\Documents\Myfile" then the file "MyFile" will be initially displayed in the 'Name' selector box, the "Documents" folder will be intially displayed in the 'Folder' selector box and "C" will be displayed in the disk selector. You must always seed the dialog: you cannot pass a null seedFile$ string to OPENFILEDIALOG$:. If seedFile$ does not include a drive name, then an error is raised.

The selection of files may be restricted by UID by specifying appropriate values for uid1&, uid2& and uid3& in exactly the same way as when using the dFILE keyword. See the 'General Changed Keywords' section for details of this.

The return value is the filename, including the full path of the file selected to be opened.

CREATEFILEDIALOG$:

Usage: `file$=CREATEFILEDIALOG$:(seedPath$)`

This procedure displays the standard 'Create new file' dialog. seedPath$ provides a starting path which is displayed when the dialog is opened. So, for example, if seedPath$="C:\Documents\" then the "Documents" folder will be intially displayed in the 'Folder' selector box and "C" in the disk selector. If you supply a filename in seedPath$, then this will be displayed as a suggested filename in the 'Name' selector box. You must always seed the dialog: you cannot pass a null seedPath$ string to CREATEFILEDIALOG$:. If seedPath$ does not include a drive name, then an error is raised.

The return value is the filename, including the full path of the file to be created.

SAVEASFILEDIALOG$:

Usage: `path$=SAVEASFILEDIALOG$:(seedPath$,BYREF useNewFile%)`

This procedure displays the standard 'Save as' dialog. seedPath$ provides a starting path which is displayed when the dialog is opened. So, for example, if seedPath$="C:\Documents\" then the "Documents" folder will be intially displayed in the 'Folder' selector box and "C" in the disk selector. If you supply a filename in seedPath$, then this will be displayed as a suggested filename in the 'Name' selector box. You must always seed the dialog: you cannot pass a null seedPath$ string to SAVEASFILEDIALOG$:. If seedPath$ does not include a drive name, then an error is raised.

UseNewFile% specifies the initial setting of the checkbox which determines whether a new file should be used when saving. This value is non-zero then the tick will be set initially, if it is zero, then it will not be. The procedure writes back a value to this variable which will be KTrue% if the symbol is set on exiting the dialog and KFalse% if not (see Const.oph for definitions of these constants). If you pass #0 as the value of this argument, then this item will not be displayed in the dialog at all (as in many Series 5 applications) so that whether a new file is used on not is not decided by the user, but by the programmer.

The return value is the filename to save as, including the full path.

SPRITE AND BITMAP OPX

In general sprite procedures behave in a similar way to the sprite keywords of OPL16.

To use this OPX, you must included the header file Bmp.oxh, which contains the following declaration:

```
DECLARE OPX BMP,$10
   BITMAPLOAD&:(name$,index&)
   BITMAPUNLOAD:(id&)
   BITMAPDISPLAYMODE&:(id&)
   SPRITECREATE&:(winId%,x&,y&,flags&)
   SPRITEAPPEND:(time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)
   SPRITECHANGE:(id&,time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)
   SPRITEDRAW:()
   SPRITEPOS:(x&,y&)
   SPRITEDELETE:(id&)
   SPRITEUSE:(id&)
END DECLARE
```

BITMAPLOAD&:

Usage: `id&=BITMAPLOAD&:(name$,index&)`

This procedure loads a bitmap from the file name$ (in the current directory if no other is specified). If the file contains more than one bitmap, then index& specifies which bitmap to load. The first (or only) bitmap is specified by index& having the value 0. The value returned is the ID of the open bitmap, which may be used to refer to it when calling sprite procedures or using the gBUTTON function.

See BITMAPUNLOAD:.

BITMAPUNLOAD:

Usage: `BITMAPUNLOAD:(id&)`

This procedure unloads the bitmap whose ID is id&. You can call BITMAPUNLOAD: immediately after passing the bitmap ID to gBUTTON or, if used with the Sprite OPX procedures, **after** drawing the sprite, as the access count on a bitmap is incremented to ensure it is not unloaded prematurely.

See BITMAPLOAD&:.

BITMAPDISPLAYMODE&:

Usage:`mode&=BITMAPDISPLAYMODE&:(id&)`

This procedure returns the graphics mode of the bitmap with ID id&. The graphics mode is that specified at its creation (see gCREATEBIT), i.e. 0 for 2-colour mode, 1 for 4-colour mode and 2 for 16-colour mode.

SPRITECREATE&:

Usage: `id&=SPRITECREATE&:(winId%,x&,y&,flags&)`

This procedure initialises a sprite in the window given by winId% with its top left-hand corner at x&,y&. The value of flags& determines whether or not the sprite's members are flashing. If (flags& AND 1)=1 then they are flashing and if the value is 0 then they are not. The value returned is the ID of the sprite which should be used when referring to it in other sprite procedures.

See SPRITEAPPEND:, SPRITECHANGE:, SPRITEDRAW:, SPRITEPOS:, SPRITEDELETE:, SPRITEUSE:.

SPRITEAPPEND:

Usage: `SPRITEAPPEND:(time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)`

This procedure is used to append *members* or *frames* to the current sprite, which must have been created using SPRITECREATE&: before attempting to append to it.

bitmap& is the ID of the bitmap to be used for this member of the sprite, and maskBitmap& is the ID of the bitmap to be used as a mask. The bitmap and mask must be of identical sizes. invertMask& takes the value of 1 or 0 to determine whether the mask is to be inverted or not respectively. For example, if you are using identical bitmap and mask and invertMask&=1 then the member will appear as the bitmap, whereas if invertMask&=0, the member will be blank. time& is the period of time in microseconds for which the member appears in the sprite and dx&,dy& is the offset position of the top left-hand of the bitmap from the top left-hand corner of the sprite.

See SPRITECHANGE:.

SPRITECHANGE:

Usage: `SPRITECHANGE:(id&,time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)`

This procedure may be used to change a member of a sprite originally set up with SPRITEAPPEND:.

id& specifies the number of the sprite member which is to be changed. This number is determined by the order in which the members were originally appended, the first member to be appended being labelled 0. bitmap& is the ID of the bitmap to be used for this member of the sprite, and maskBitmap& is the ID of the bitmap to be used as a mask. The bitmap and mask must be of identical sizes. invertMask& takes the value of 1 or 0 to determine whether the mask is to be inverted or not respectively. For example, if you are using identical bitmap and mask and invertMask&=1 then the member will appear as the bitmap, whereas if invertMask&=0, the member will be blank. time& is the period of time in microseconds for which the member appears in the sprite and dx&,dy& is the offset position of the top left-hand of the bitmap from the top left-hand corner of the sprite.

A sprite may not be changed unless it has already been drawn.

See SPRITEAPPEND:, SPRITEDRAW:.

SPRITEDRAW:

Usage: `SPRITEDRAW:`

This procedure draws the sprite once it has been set up. It need only be called once for each sprite, and any changes made to the sprite are made as soon as the procedure making the change is called. The sprite will be drawn in the window and at the position specified by the arguments passed to SPRITECREATE:.

See SPRITECREATE&:, SPRITEAPPEND:, SPRITECHANGE:, SPRITEPOS:.

SPRITEPOS:

Usage: `SPRITEPOS:(x&,y&)`

This procedure may be used to reposition the whole of the current sprite to the point x&,y&.

See SPRITECREATE&:,SPRITEDRAW:.

SPRITEDELETE:

Usage: `SPRITEDELETE:(id&)`

This procedure deletes the sprite with ID id&.

SPRITEUSE:

Usage: `SPRITEUSE:(id&)`

This procedure sets the current sprite to be that with ID id& in order that it may be changed.

See SPRITECHANGE:, SPRITEPOS:.

Note that using many sprites at one time may cause undesirable effects. Firstly, sprites maybe animated at a slower speed than requested, and secondly the response time to any key or pointer event can be lengthened considerably. The number of sprites which can be drawn at one time without these effects occurring depends mainly on the animation speed: the faster it is the less sprites may be used successfully at one time. As a general rule, it is advisable not to use more than around 20 sprites at any one time when the animation speed is 0.2 secs, but you should experiment to find out what is suitable for your particular program.

DATABASE OPX

To use this OPX, you must included the header file Dbase.oxh, which contains the following declaration:

```
DECLARE OPX DBASE,KUidOpxDBase&,KOpxDBaseVersion%
   DbAddField:(keyPtr&,fieldName$,order&) : 1
   DbAddFieldTrunc:(keyPtr&,fieldName$,order&,trunc&) : 2
   DbCreateIndex:(index$,keyPtr&,dbase$,table$) : 3
   DbDeleteKey:(keyPtr&) : 4
   DbDropIndex:(index$,dbase$,table$) : 5
   DbGetFieldCount&:(dbase$,table$) : 6
   DbGetFieldName$:(dbase$,table$,fieldNum&) : 7
   DbGetFieldType&:(dbase$,table$,fieldNum&) : 8
   DbIsDamaged&:(dbase$) : 9
   DbIsUnique&:(keyPtr&) : 10
   DbMakeUnique:(keyPtr&) : 11
   DbNewKey&: : 12
   DbRecover:(dbase$) : 13
   DbSetComparison:(KeyPtr&,comp&) : 14
END DECLARE
```

DBADDFIELD:

Usage: `DBADDFIELD:(key&,field$,order&)`

This procedure adds the field field$ to the *key* key& (see DBNEWKEY: to create a key). This new key can then be used to create an *index* on a table. order& specifies how this field should be ordered. The following two constants are supplied in Dbase.oxh for specifying ascending and descending order respectively:

```
CONST KDbAscending&=1
CONST KDbDescending&=0
```

You can use this procedure repeatedly to add more than one field to the same key. The order in which the fields are added dictates the significance of these fields in building up the index. The first field added to a key is the primary field; if there were any identical values in this field then the secondary field would be used and so on. For example, if the primary field in an 'Address' table was 'Surname' and there were two people with the surname Brown, then the secondary field or tertiary fields like 'Forename' or 'Age' might be used to define how the index will be ordered.

When using string fields in an index they cannot be longer than 240 bytes. To limit the size of a string field see CREATE. If a string field is the last field in a key (or first and only) then it may be truncated to a specified length - see DBADDFIELDTRUNC:.

Note that the size of an index can become huge if the key is long. The key length *k* is the length in bytes of the sum of all the fields in the key. An integer or a long integer field is 4 bytes, a floating-point field 8 bytes and a text field depends on the length assigned to it (see also the previous paragraph). The size of the index depends at least linearly on *k* and hence may be large if the key is long.

When the key has been built as required it can be used to create an index.

See DBCREATEINDEX:.

DBADDFIELDTRUNC:

Usage: `DBADDFIELDTRUNC:(key&,field$,order&,trunc&)`

This procedure adds a truncated string field field$ to the key key&. Only the last field added to a key (or first and only) can be truncated. order& specifies how this field should be ordered. The following two constants are supplied in Dbase.oxh for specifying ascending and descending order respectively:

```
CONST KDbAscending&=1
CONST KDbDescending&=0
```

trunc& is the truncation length and determines how many bytes of the string field will be used in building up the index (up to 240).

This procedure is useful for building up indexes on OPL16 databases whose string fields have a set length of 255 bytes. When the key has been built up as required it can be used to create an index.

See DBADDFIELD:, DBCREATEINDEX:.

DBCREATEINDEX:

Usage: `DBCREATEINDEX:(index$,key&,file$,table$)`

This creates an index with the name index$ on the table table$ in the database file$. The index is used for sorting and vastly increases the speed in table lookup. The index is based on key&, the supplied key. The database must be closed when this procedure is used.

The opening of an ordered view on a table must be requested in the SQL query in the OPEN command. If a suitable index has been created then it will be used.

See DBNEWKEY:, DBADDFIELD:, DBADDFIELDTRUNC:, DBMAKEUNIQUE:, DBSETCOMPARISON:, DBDROPINDEX:, OPEN.

DBDELETEKEY:

Usage: `DBDELETEKEY:(key&)`

This deletes the key key&. This should be called as soon as the key is no longer required. Any keys left undeleted when all modules that declare or include a declaration of the Dbase OPX have been unloaded will be automatically deleted

See DBNEWKEY&:.

DBDROPINDEX:

Usage: `DBDROPINDEX:(index$,file$,table$)`

This drops the index index$ on the table table$ of the database file$. The database must be closed to use this procedure.

See DBCREATEINDEX:.

DBGETFIELDCOUNT&:

Usage: `n&=DBGETFIELDCOUNT&:(dbase$,table$)`

This procedure returns the number of fields in one of the records in the table table$ of the database dbase$. This number can then be used to analyse the contents of the record.

See DBGETFIELDNAME$:, DBGETFIELDTYPE&:

DBGETFIELDNAME$:

Usage: `name$=DBGETFIELDNAME$:(dbase$,table$,fieldNum&)`

This procedure returns the name of the field whose number is fieldNum& in the table table$ of the database dbase$. This is useful for analysing the records of a table.

See DBGETFIELDCOUNT&:, DBGETFIELDTYPE&:.

DBGETFIELDTYPE&:

Usage: `type&=DBGETFIELDTYPE&:(dbase$,table$,fieldNum&)`

This procedure returns the type of the field whose number is fieldNum& in the table table$ of the database dbase$. This is useful for analysing the records of a table.

The values that may be returned (many of which aren't supported by OPL databases) are as follows:

value	*type*
0	bit
1	signed byte (8 bits)
1	unsigned byte (8 bits)
2	integer (16 bits)
3	unsigned integer (16 bits)
4	long integer (32 bits)
5	unsigned long integer (32 bits)
6	64-bit integer
7	single precision floating-point number (32 bits)
8	double precision floating-point number (64 bits)
9	date/time object
10	ASCII text
11	Unicode text
12	Binary
13	LongText8
14	LongText16
15	LongBinary

Types 2,4,8,10 correspond to OPL's %,&, floating point and $ types respectively.

See DBGETFIELDCOUNT&, DBGETFIELDNAME&

DBISDAMAGED&:

Usage: `i&=DBISDAMAGED&:(dbase$)`

This procedure returns 1 if the database dbase$ considers that it may have damaged indexes and 0 if not. If the database is damaged, then this does not make it unusable, but attempting to use any damaged index will result in an error. Recovering the database will restore any damaged indexes.

See DBRECOVER:.

DBISUNIQUE&:

Usage: `i&=DBISUNIQUE&:(key&)`

This procedure returns -1 if the key key& is unique or 0 if it is not.

See DBMAKEUNIQUE:.

DBMAKEUNIQUE:

Usage: `DBMAKEUNIQUE:(key&)`

This sets the key key& to be unique. The index created will then not allow any records to be added if they exactly match the indexed fields of another record.

See DBNEWKEY:, DBADDFIELD:, DBCREATEINDEX:.

DBNEWKEY&:

Usage: `k&=NEWKEY&:`

This procedures returns a handle to a key which can be used for creating indexes.

See DBDELETEKEY:, DBCREATEINDEX:, DBADDFIELD:, DBADDFIELDTRUNC:.

DBRECOVER:

Usage: `DbRecover:(dbase$)`

This recovers a damaged database dbase$, restoring any damaged indices.

See DBISDAMAGED&:.

DBSETCOMPARISON:

Usage: `DBSETCOMPARISON:(key&,comp&)`

The text comparison is used to determine the order of an index. The argument comp& sets the text comparsion mode of a key key&, and takes one of the constant values are supplied in Dbase.oxh:

```
CONST KDbCompareNormal&=     0
CONST KDbCompareFolded&=     1
CONST KDbCompareCollated&=   2
```

PRINTER OPX

Printer OPX provides access to print and text handling. An object to be printed is first created dynamically by sending text, bitmaps and formatting information. Printing itself is then handled by calling standard print dialogs. One procedure is provided for each of the four dialogs usually called from standard applications.

To use this OPX, you must included the header file Printer.oxh, which contains the following declaration:

```
DECLARE OPX PRINTER,KUidOpxPrinter&,KOpxPrinterVersion%
   SendStringToPrinter:(string$) : 1
   InsertString:(string$,pos&) : 2
   SendNewParaToPrinter: : 3
   InsertNewPara:(pos&) : 4
   SendSpecialCharToPrinter:(character%) : 5
   InsertSpecialChar:(character%,pos&) : 6
   SetAlignment:(alignment%) : 7
```

```
    InitialiseParaFormat:(Red&, Green&, Blue&, LeftMarginInTwips&,
    RightMarginInTwips&, IndentInTwips&, HorizontalAlignment%,
    VerticalAlignment%, LineSpacingInTwips&, LineSpacingControl%,
    SpaceBeforeInTwips&, SpaceAfterInTwips&, KeepTogether%, KeepWithNext%,
    StartNewPage%, WidowOrphan%, Wrap%, BorderMarginInTwips&,
    DefaultTabWidthInTwips&) : 8


    SetLocalParaFormat: : 9
    SetGlobalParaFormat: : 10
    RemoveSpecificParaFormat: : 11
    SetFontName:(name$) : 12
    SetFontHeight:(height%) : 13
    SetFontPosition:(pos%) : 14
    SetFontWeight:(weight%) : 15
    SetFontPosture:(posture%) : 16
    SetFontStrikethrough:(strikethrough%) : 17
    SetFontUnderline:(underline%) : 18
    SetGlobalCharFormat: : 19
    RemoveSpecificCharFormat: : 20
    SendBitmapToPrinter:(bitmapHandle&) : 21
    InsertBitmap:(bitmapHandle&,pos&) : 22
    SendScaledBitmapToPrinter:(bitmapHandle&,xScale&,yScale&) : 23
    InsertScaledBitmap:(bitmapHandle&,pos&,xScale&,yScale&) : 24
    PrinterDocLength&: : 25
    SendRichTextToPrinter:(richTextAddress&) : 26
    ResetPrinting: : 27
    PageSetupDialog: : 28
    PrintPreviewDialog: : 29
    PrintRangeDialog: : 30
    PrintDialog: : 31
    SendBufferToPrinter:(Addr&) : 32
END DECLARE
```

SENDSTRINGTOPRINTER:

Usage: `SENDSTRINGTOPRINTER:(string$)`

Append the string string$ to the buffer which has already been sent to the printer.

See INSERTSTRING:, SENDNEWPARATOPRINTER:.

INSERTSTRING:

Usage: `INSERTSTRING:(string$,pos&)`

Inserts string$ at position pos$ in the buffer. Inserting at position zero puts the string at the beginning of the buffer already sent.

See SENDSTRINGTOPRINTER:, INSERTNEWPARA:.

SENDNEWPARATOPRINTER:

Usage: `SENDNEWPARATOPRINTER:`

Sends a new paragraph to the printer. Paragraphs delimit paragraph formatting, such as centering. This procedure is equivalent to `SENDSPECIALCHARTOPRINTER:(KParagraphDelimiter%)`

See SENDSPECIALCHARTOPRINTER:, INSERTNEWPARA:, SENDSTRINGTOPRINTER:.

INSERTNEWPARA:

Usage: `INSERTNEWPARA:(pos&)`

Inserts a new paragraph at position pos& in the buffer

See SENDNEWPARATOPRINTER:, INSERTSTRING:.

SENDSPECIALCHARTOPRINTER:

Usage: `SENDSPECIALCHARTOPRINTER:(character%)`

Sends a special character, for example, line and page breaks etc. to the printer. These constant definitions supplied in Const.oph may be used:

```
CONST KParagraphDelimiter%      =$06
CONST KLineBreak%               =$07
CONST KPageBreak%               =$08
CONST KTabCharacter%            =$09
CONST KNonBreakingTab%          =$0a
CONST KNonBreakingHyphen%       =$0b
CONST KPotentialHyphen%         =$0c
CONST KNonBreakingSpace%        =$10
CONST KVisibleSpaceCharacter%   =$0f
```

See SENDNEWPARATOPRINTER:, INSERTSPECIALCHAR:.

INSERTSPECIALCHAR:

Usage: `INSERTSPECIALCHAR:(character%,pos&)`

Inserts a special character at position pos& in the buffer.

See SENDSPECIALCHARTOPRINTER:.

SETALIGNMENT:

Usage: `SETALIGNMENT:(alignment%)`

Sets paragraph alignment state. The default is KPrintLeftAlign%.

The setting does not take effect until either SETLOCALPARAFORMAT: or SETGLOBALPARAFORMAT: is called. The allowable alignments, defined in Printer.oxh, are:

```
CONST KPrintLeftAlign%          = 0
CONST KPrintCenterAlign%        = 1
CONST KPrintRightAlign%         = 2
CONST KPrintJustifiedAlign%     = 3
CONST KPrintUnspecifiedAlign%   = 4
```

See INITIALISEPARAFORMAT:.

INITIALISEPARAFORMAT:

Usage: `INITIALISEPARAFORMAT:(Red%, Green%, Blue%, LeftMarginInTwips&, RightMarginInTwips&, IndentInTwips&, HorizontalAlignment%, VerticalAlignment%, LineSpacingInTwips&, LineSpacingControl%, SpaceBeforeInTwips&, SpaceAfterInTwips&, KeepTogether%, KeepWithNext%, StartNewPage%, WidowOrphan%, Wrap%, BorderMarginInTwips&, DefaultTabWidthInTwips&)`

Sets a state for formatting. The setting does not take effect until either SETLOCALPARAFORMAT or SETGLOBALPARAFORMAT is called.

Red%,Green%,Blue% sets the background colour. Each value can be in the range 0 to 255. Default colour is white (all three equal to 255).

LeftMarginInTwips& sets left text margin relative to left page margin. Default is zero.

RightMarginInTwips& sets right text margin relative to right page margin. Default is zero.

IndentInTwips& sets left, right and first line indent. Default is zero.

HorizontalAlignment% sets horizontal alignment of paragraph. Default is KPrintLeftAlign%. See SETALIGNMENT:.

VerticalAlignment% sets vertical alignment of paragraph, for use by a spreadsheet application. Default is KPrintTopAlign%. Allowable values, supplied as constants in Printer.oxh, are:

```
CONST KPrintTopAlign%            = 0
CONST KPrintBottomAlign%         = 2
CONST KPrintUnspecifiedAlign%    = 4
```

LineSpacingInTwips& sets inter-line spacing in twips. Default is 200 (10 point).

LineSpacingControl% sets control for LineSpacingInTwips& value. Default is KLineSpacingAtLeastInTwips&. Allowable values, supplied as constants in Printer.oxh, are:

```
CONST KLineSpacingAtLeastInTwips% = 0
CONST KLineSpacingExactlyInTwips% = 1
```

SpaceBeforeInTwips& sets space above paragraph. Default is zero.

SpaceAfterInTwips& sets space below paragraph. Default is zero.

KeepTogether% prevents a page break within a paragraph when equal to KTrue%. Default is KFalse%. (These constants are supplied in Const.oph.)

KeepWithNext% prevents a page break between this paragraph and the following paragraph when equal to KTrue%. Default is KFalse%.

StartNewPage% inserts a page break immediately before this paragraph when equal to KTrue%. Default is KFalse%.

WidowOrphan% prevents the printing of the last line of a paragraph by itself on the top of a new page (widow) or the first line of a paragraph by itself on the bottom of a page (orphan) when equal to KTrue%. Default is KFalse%.

Wrap% forces the paragraph to line wrap at the right margin when equal to KTrue%; disables line wrap when equal to KFalse%. Default is KTrue%

BorderMarginInTwips& sets distance in twips between paragraph border and enclosed text (must be non-negative). Default is zero.

DefaultTabWidthInTwips& sets the spacing between the default tab stops. Default is 360.

SETLOCALPARAFORMAT:

Usage: `SETLOCALPARAFORMAT:`

Set the initialised paragraph formatting as local.

See SETGLOBALPARAFORMAT:.

SETGLOBALPARAFORMAT:

Usage: `SETGLOBALPARAFORMAT:`

Sets the initialised paragraph formatting as global.

See SETLOCALPARAFORMAT:, REMOVESPECIFICPARAFORMAT:.

REMOVESPECIFICPARAFORMAT:

Usage: `REMOVESPECIFICPARAFORMAT:`

Unsets local paragraph formatting, using the global formatting instead.

See SETGLOBALPARAFORMAT:, SETLOCALPARAFORMAT:.

SETFONTNAME:

Usage: `SETFONTNAME:(name$)`

Sets the font typeface which will be used for subsequent printing to name$. name$ can be “Arial” (san serif font), “Times New Roman” (font with serifs) or “Courier New” (monospaced font). The default font is Courier New.

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTHEIGHT:

Usage: `SETFONTHEIGHT:(height%)`

Sets a new font height in twips:

1 twip = 1/20 point or 1/1440 inch.

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTPOSITION:

Usage: `SETFONTPOSITION:(pos%)`

Sets font position as one of the following (constants are supplied in Printer.oxh):

```
CONST KPrintPosNormal%       = 0
CONST KPrintPosSuperscript%  = 1
CONST KPrintPosSubscript%    = 2
```

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTWEIGHT:

Usage: `SETFONTWEIGHT:(weight%)`

Sets the font weight (bold or normal). Allowable values are:

```
CONST KStrokeWeightNormal%   = 0
CONST KStrokeWeightBold%     = 1
```

(Constants are supplied in Printer.oxh.)

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTPOSTURE:

Usage: `SETFONTPOSTURE:(posture%)`

Sets the font posture (upright or italic). Allowable values are:

```
CONST KPostureUpright%       = 0
CONST KPostureItalic%        = 1
```

(Constants are supplied in Printer.oxh.)

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTSTRIKETHROUGH:

Usage: `SETFONTSTRIKETHROUGH:(strikethrough%)`

Sets strikethrough on or off. Allowable values are:

```
CONST KStrikethroughOff%     = 0
CONST KStrikethroughOn%      = 1
```

(Constants are supplied in Printer.oxh.)

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETFONTUNDERLINE:

Usage: `SETFONTUNDERLINE:(underline%)`

Sets underlining on or off. Allowable values are:

```
CONST KUnderlineOff%    = 0
CONST KUnderlineOn%     = 1
```

(Constants are supplied in Printer.oxh.)

This takes immediate effect locally, but requires the use of SETGLOBALCHARFORMAT to set globally.

SETGLOBALCHARFORMAT:

Usage: `SETGLOBALCHARFORMAT:`

Sets the currently active character format to be the global default.

See REMOVESPECIFICCHARFORMAT:.

REMOVESPECIFICCHARFORMAT:

Usage: `REMOVESPECIFICCHARFORMAT:`

Unsets the local character formatting, using the global instead.

See SETGLOBALCHARFORMAT:.

SENDBITMAPTOPRINTER:

Usage: `SENDBITMAPTOPRINTER:(bitmapHandle&)`

Appends a bitmap from its handle bitmapHandle& to the buffer sent to the printer. The handle is the same as that returned by gLOADBIT.

See INSERTBITMAP:, SENDSCALEDBITMAPTOPRINTER:.

INSERTBITMAP:

Usage: `INSERTBITMAP:(bitmapHandle&,pos&)`

Inserts a bitmap specified by bitmapHandle& at position pos& in the buffer to be sent to the printer. The handle is the same as that returned by gLOADBIT.

See SENDBITMAPTOPRINTER:, INSERTSCALEDBITMAP:.

SENDSCALEDBITMAPTOPRINTER:

Usage: `SENDSCALEDBITMAPTOPRINTER:(bitmapHandle&,xScale&,yScale&)`

Appends a scaled bitmap specified by bitmapHandle& and scaled according to xScale& and yScale& to the buffer sent to the printer. The handle is the same as that returned by gLOADBIT. Default scaling is xScale& = yScale& = 1000.

See SENDBITMAPTOPRINTER:, INSERTSCALEDBITMAP:.

INSERTSCALEDBITMAP:

Usage: `INSERTSCALEDBITMAP:(bitmapHandle&,pos&,xScale&,yScale&)`

Inserts a scaled bitmap specified by bitmapHandle& and scaled according to xScale& and yScale& at position pos& in the buffer sent to the printer. The handle is the same as that returned by gLOADBIT.

See INSERTBITMAP:, SENDSCALEDBITMAPTOPRINTER:.

PRINTERDOCLENGTH&

Usage: `length&=PRINTERDOCLENGTH&:`

Returns the count of characters currently held in the buffer. It is not possible to insert characters beyond the length of the document buffer. Bitmaps and special characters each count as one character.

SENDRICHTEXTTOPRINTER:

Usage: `SENDRICHTEXTTOPRINTER:(richTextAddress&)`

Sends the address of a rich text object richTextAddress& to the printer. This is intended to be used with a pointer returned by another OPX. This new rich text will **replace** all contents currently stored.

RESETPRINTING:

Usage: `RESETPRINTING:`

Deletes **all** text, bitmaps and formatting in the printing buffer.

PAGESETUPDIALOG:

Usage: `PAGESETUPDIALOG:`

Calls the standard page setup dialog.

See PRINTPREVIEWDIALOG:, PRINTRANGEDIALOG:, PRINTDIALOG:.

PRINTPREVIEWDIALOG:

Usage: `PRINTPREVIEWDIALOG:`

Calls the standard print preview dialog. This allows the data sent to the printer to be viewed. The other three associated dialogs can all be called from this dialog.

See PAGESETUPDIALOG:, PRINTRANGEDIALOG:, PRINTDIALOG:.

PRINTRANGEDIALOG:

Usage: `PRINTRANGEDIALOG:`

Calls the standard print range dialog.

See PAGESETUPDIALOG:, PRINTPREVIEWDIALOG:, PRINTDIALOG:.

PRINTDIALOG:

Usage: `PRINTDIALOG:`

Calls the standard print dialog.

See PAGESETUPDIALOG:, PRINTPREVIEWDIALOG:, PRINTRANGEDIALOG:.

SENDBUFFERTOPRINTER:

Usage: `SENDBUFFERTOPRINTER:(addr&)`

This procedure appends the contents of a buffer to whatever has already been sent to the printer. The addr& parameter should be the address of a buffer into which text has been inserted by dEDITMULTI.

APPENDIX A

CONST.OPH

The following is the content of CONST.OPH.

```
REM CONST.OPH version 1.01
REM Constants for OPL
REM Last edited on 10 July 1997
REM (C) Copyright Psion PLC 1997

REM Changed in 1.01:
REM - some language codes added
REM - consts for KKeyUpArrow%,KKeyDownArrow%,... apply to 16-bit
REM   keywords GET, GETEVENT, KEY
REM   KKeyUpArrow32%,KKeyDownArrow32%,... added for 32-bit
REM   keywords GETEVENT32 etc.

rem General constants
const KTrue%=-1
const KFalse%=0

rem Data type ranges
const KMaxStringLen%=255
const KMaxFloat=1.7976931348623157E+308
const KMinFloat=2.2250738585072015E-308
                  rem Minimum with full precision in mantissa
const KMinFloatDenorm=5e-324
                  rem Denormalised (just one bit of precision left)
const KMinInt%=$8000
                  rem -32768 (translator needs hex for maximum ints)
const KMaxInt%=32767
const KMinLong&=&80000000    rem -2147483648 (hex for translator)
const KMaxLong&=2147483647

rem Special keys
const KKeyEsc%=27
const KKeySpace%=32
const KKeyDel%=8
const KKeyTab%=9
const KKeyEnter%=13
rem Key constants for 16-bit keywords GETEVENT etc.
const KGetMenu%=290
const KKeyUpArrow%=256
const KKeyDownArrow%=257
const KKeyLeftArrow%=259
const KKeyRightArrow%=258
const KKeyPageUp%=260
const KKeyPageDown%=261
const KKeyPageLeft%=262
const KKeyPageRight%=263
const KKeyMenu%=4150              rem const kept for compatibility
const KKeySidebarMenu%=10000 rem const kept for compatibility
rem Key constants for 32-bit keywords GETEVENT32 etc.
const KKeyMenu32%=4150
const KKeySidebarMenu32%=10000
```

```
const KKeyPageLeft32%=4098
const KKeyPageRight32%=4099
const KKeyPageUp32%=4100
const KKeyPageDown32%=4101
const KKeyLeftArrow32%=4103
const KKeyRightArrow32%=4104
const KKeyUpArrow32%=4105
const KKeyDownArrow32%=4106
rem Month numbers
const KJanuary%=1
const KFebruary%=2
const KMarch%=3
const KApril%=4
const KMay%=5
const KJune%=6
const KJuly%=7
const KAugust%=8
const KSeptember%=9
const KOctober%=10
const KNovember%=11
const KDecember%=12

rem Graphics
const KDefaultWin%=1
const KgModeSet%=0
const KgModeClear%=1
const KgModeInvert%=2
const KtModeSet%=0
const KtModeClear%=1
const KtModeInvert%=2
const KtModeReplace%=3
const KgStyleNormal%=0
const KgStyleBold%=1
const KgStyleUnder%=2
const KgStyleInverse%=4
const KgStyleDoubleHeight%=8
const KgStyleMonoFont%=16
const KgStyleItalic%=32

rem For 32-bit status words IOWAIT and IOWAITSTAT32
rem Use KErrFilePending% (-46) for 16-bit status words
const KStatusPending32&=&80000001

rem Error codes
const KErrGenFail%=-1
const KErrInvalidArgs%=-2
const KErrOs%=-3
const KErrNotSupported%=-4
const KErrUnderflow%=-5
const KErrOverflow%=-6
const KErrOutOfRange%=-7
const KErrDivideByZero%=-8
const KErrInUse%=-9
const KErrNoMemory%=-10
const KErrNoSegments%=-11
const KErrNoSemaphore%=-12
const KErrNoProcess%=-1
```

```
const KErrAlreadyOpen%=-14
const KErrNotOpen%=-15
const KErrImage%=-16
const KErrNoReceiver%=-17
const KErrNoDevices%=-18
const KErrNoFileSystem%=-19
const KErrFailedToStart%=-20
const KErrFontNotLoaded%=-21
const KErrTooWide%=-22
const KErrTooManyItems%=-23
const KErrBatLowSound%=-24
const KErrBatLowFlash%=-25
const KErrExists%=-32
const KErrNotExists%=-33
const KErrWrite%=-34
const KErrRead%=-35
const KErrEof%=-36
const KErrFull%=-37
const KErrName%=-38
const KErrAccess%=-39
const KErrLocked%=-40
const KErrDevNotExist%=-41
const KErrDir%=-42
const KErrRecord%=-43
const KErrReadOnly%=-44
const KErrInvalidIO%=-45
const KErrFilePending%=-46
const KErrVolume%=-47
const KErrIOCancelled%=-48
rem OPL specific errors
const KErrSyntax%=-77
const KOplStructure%=-85
const KErrIllegal%=-96
const KErrNumArg%=-97
const KErrUndef%=-98
const KErrNoProc%=-99
const KErrNoFld%=-100
const KErrOpen%=-101
const KErrClosed%=-102
const KErrRecSize%=-103
const KErrModLoad%=-104
const KErrMaxLoad%=-105
const KErrNoMod%=-106
const KErrNewVer%=-107
const KErrModNotLoaded%=-108
const KErrBadFileType%=-109
const KErrTypeViol%=-110
const KErrSubs%=-111
const KErrStrTooLong%=-112
const KErrDevOpen%=-113
const KErrEsc%=-114
const KErrMaxDraw%=-117
const KErrDrawNotOpen%=-118
const KErrInvalidWindow%=-119
const KErrScreenDenied%=-120
const KErrOpxNotFound%=-121
```

```
const KErrOpxVersion%=-122
const KErrOpxProcNotFound%=-123
const KErrStopInCallback%=-124
const KErrIncompUpdateMode%=-125
const KErrInTransaction%=-126

rem For ALERT
const KAlertEsc%=1
const KAlertEnter%=2
const KAlertSpace%=3

rem For BUSY and GIPRINT
const KBusyTopLeft%=0
const KBusyBottomLeft%=1
const KBusyTopRight%=2
const KBusyBottomRight%=3
const KBusyMaxText%=80

rem For CMD$
const KCmdAppName%=1   rem Full path name used to start program
const KCmdUsedFile%=2
const KCmdLetter%=3
rem For CMD$(3)
const KCmdLetterCreate$="C"
const KCmdLetterOpen$="O"
const KCmdLetterRun$="R"

rem For CURSOR
const KCursorTypeNotFlashing%=2
const KCursorTypeGrey%=4

rem For DATIM$ - offsets
const KDatimOffDayName%=1
const KDatimOffDay%=5
const KDatimOffMonth%=8
const KDatimOffYear%=12
const KDatimOffHour%=17
const KDatimOffMinute%=20
const KDatimOffSecond%=23

rem For dBUTTON
const KDButtonNoLabel%=$100
const KDButtonPlainKey%=$200
const KDButtonDel%=8
const KDButtonTab%=9
const KDButtonEnter%=13
const KDButtonEsc%=27
const KDButtonSpace%=32

rem For dEDITMULTI and printing
const KParagraphDelimiter%=$06
const KLineBreak%=$07
const KPageBreak%=$08
const KTabCharacter%=$09
const KNonBreakingTab%=$0a
const KNonBreakingHyphen%=$0b
```

```
const KPotentialHyphen%=$0c
const KNonBreakingSpace%=$10
const KPictureCharacter%=$0e
const KVisibleSpaceCharacter%=$0f

rem For DEFAULTWIN
const KDefWin4ColourMode%=1
const KDefWin16ColourMode%=2

rem For dFILE
const KDFileNameLen%=255

      rem flags
const KDFileEditBox%=$0001
const KDFileAllowFolders%=$0002
const KDFileFoldersOnly%=$0004
const KDFileEditorDisallowExisting%=$0008
const KDFileEditorQueryExisting%=$0010
const KDFileAllowNullStrings%=$0020
const KDFileAllowWildCards%=$0080
const KDFileSelectorWithRom%=$0100
const KDFileSelectorWithSystem%=$0200

rem Opl-related Uids for dFILE
const KUidOplInterpreter&=268435575
const KUidOplApp&=268435572
const KUidOplDoc&=268435573
const KUidOPO&=268435571
const KUidOplFile&=268435594
const KUidOpxDll&=268435549

rem For DIALOG
const KDlgCancel%=0

rem For dINIT (flags for dialogs)
const KDlgButRight%=1
const KDlgNoTitle%=2
const KDlgFillScreen%=4
const KDlgNoDrag%=8
const KDlgDensePack%=16

rem For DOW
const KMonday%=1
const KTuesday%=2
const KWednesday%=3
const KThursday%=4
const KFriday%=5
const KSaturday%=6
const KSunday%=7

rem For dPOSITION
const KDPositionLeft%=-1
const KDPositionCentre%=0
const KDPositionRight%=1
```

```
rem For dTEXT
const KDTextLeft%=0
const KDTextRight%=1
const KDTextCentre%=2
const KDTextBold%=$100       rem Ignored in Eikon
const KDTextLineBelow%=$200
const KDTextAllowSelection%=$400
const KDTextSeparator%=$800

rem For dTIME
const KDTimeAbsNoSecs%=0
const KDTimeAbsWithSecs%=1
const KDTimeDurationNoSecs%=2
const KDTimeDurationWithSecs%=3
rem Flags for dTIME (for ORing combinations)
const KDTimeWithSeconds%=1
const KDTimeDuration%=2
const KDTimeNoHours%=4
const KDTime24Hour%=8

rem For dXINPUT
const KDXInputMaxLen%=16

rem For FINDFIELD
const KFindCaseDependent%=16
const KFindBackwards%=0
const KFindForwards%=1
const KFindBackwardsFromEnd%=2
const KFindForwardsFromStart%=3

rem For FLAGS
const KFlagsAppFileBased%=1
const KFlagsAppIsHidden%=2

rem For gBORDER and gXBORDER
const KBordSglShadow%=1
const KBordSglGap%=2
const KBordDblShadow%=3
const KBordDblGap%=4
const KBordGapAllRound%=$100
const KBordRoundCorners%=$200
const KBordLosePixel%=$400

rem For gBUTTON
const KButtS3%=0
const KButtS3Raised%=0
const KButtS3Pressed%=1
const KButtS3a%=1
const KButtS3aRaised%=0
const KButtS3aSemiPressed%=1
const KButtS3aSunken%=2
const KButtS5%=2
const KButtS5Raised%=0
const KButtS5SemiPressed%=1
const KButtS5Sunken%=2
```

```
const KButtLayoutTextRightPictureLeft%=0
const KButtLayoutTextBottomPictureTop%=1
const KButtLayoutTextTopPictureBottom%=2
const KButtLayoutTextLeftPictureRight%=3
const KButtTextRight%=0
const KButtTextBottom%=1
const KButtTextTop%=2
const KButtTextLeft%=3
const KButtExcessShare%=$00
const KButtExcessToText%=$10
const KButtExcessToPicture%=$20

rem For gCLOCK
const KgClockS5System%=6
const KgClockS5Analog%=7
const KgClockS5Digital%=8
const KgClockS5LargeAnalog%=9
const KgClockS5Formatted%=11

rem For gCREATE
const KgCreateInvisible%=0
const KgCreateVisible%=1
const KgCreate2ColourMode%=$0000
const KgCreate4ColourMode%=$0001
const KgCreate16ColourMode%=$0002
const KgCreateHasShadow%=$0010

rem For GETCMD$
const KGetCmdLetterCreate$="C"
const KGetCmdLetterOpen$="O"
const KGetCmdLetterExit$="X"
const KGetCmdLetterUnknown$="U"

rem For gLOADBIT
const KgLoadBitReadOnly%=0
const KgLoadBitWriteable%=1

rem For gRANK
const KgRankForeground%=1
const KgRankBackGround%=32767

rem For gPOLY - array subscripts
const KgPolyAStartX%=1
const KgPolyAStartY%=2
const KgPolyANumPairs%=3
const KgPolyANumDx1%=4
const KgPolyANumDy1%=5

rem For gPRINTB
const KgPrintBRightAligned%=1
const KgPrintBLeftAligned%=2
const KgPrintBCentredAligned%=3
rem The defaults
const KgPrintBDefAligned%=KgPrintBLeftAligned%
const KgPrintBDefTop%=0
const KgPrintBDefBottom%=0
const KgPrintBDefMargin%=0
```

```
rem For gXBORDER
const KgXBorderS3Type%=0
const KgXBorderS3aType%=1
const KgXBorderS5Type%=2

rem For gXPRINT
const KgXPrintNormal%=0
const KgXPrintInverse%=1
const KgXPrintInverseRound%=2
const KgXPrintThinInverse%=3
const KgXPrintThinInverseRound%=4
const KgXPrintUnderlined%=5
const KgXPrintThinUnderlined%=6

rem For KMOD
const KKmodShift%=2
const KKmodControl%=4
const KKmodPsion%=8
const KKmodCaps%=16
const KKmodFn%=32

rem For mCARD and mCASC
const KMenuDimmed%=$1000
const KMenuSymbolOn%=$2000
const KMenuSymbolIndeterminate%=$4000
const KMenuCheckBox%=$0800
const KMenuOptionStart%=$0900
const KMenuOptionMiddle%=$0A00
const KMenuOptionEnd%=$0B00

rem For mPOPUP position type
rem Specifies which corner of the popup is given by the coordinates
const KMPopupPosTopLeft%=0
const KMPopupPosTopRight%=1
const KMPopupPosBottomLeft%=2
const KMPopupPosBottomRight%=3

rem For PARSE$ - array subscripts
const KParseAOffFSys%=1
const KParseAOffDev%=2
const KParseAOffPath%=3
const KParseAOffFilename%=4
const KParseAOffExt%=5
const KParseAOffWild%=6
rem Wild-card flags
const KParseWildNone%=0
const KParseWildFilename%=1
const KParseWildExt%=2
const KParseWildBoth%=3

rem For SCREENINFO - array subscripts
const KSInfoALeft%=1
const KSInfoATop%=2
const KSInfoAScrW%=3
const KSInfoAScrH%=4
const KSInfoAReserved1%=5
```

```
const KSInfoAFont%=6
const KSInfoAPixW%=7
const KSInfoAPixH%=8
const KSInfoAReserved2%=9
const KSInfoAReserved3%=10

rem For SETFLAGS
const KRestrictTo64K&=&0001
const KAutoCompact&=&0002
const KTwoDigitExponent&=&0004
const KSendSwitchOnMessage&=&010000

rem For GetEvent32
rem Array indexes
const KEvAType%=1
const KEvATime%=2

rem event array keypress subscripts
const KEvAKMod%=4
const KEvAKRep%=5

rem Pointer event array subscripts
const KEvAPtrOplWindowId%=3
const KEvAPtrWindowId%=3
const KEvAPtrType%=4
const KEvAPtrModifiers%=5
const KEvAPtrPositionX%=6
const KEvAPtrPositionY%=7
const KEvAPtrScreenPosX%=8
const KEvAPtrScreenPosY%=9

rem Event types
const KEvNotKeyMask&=&400
const KEvFocusGained&=&401
const KEvFocusLost&=&402
const KEvSwitchOn&=&403
const KEvCommand&=&404
const KEvDateChanged&=&405
const KEvKeyDown&=&406
const KEvKeyUp&=&407
const KEvPtr&=&408
const KEvPtrEnter&=&409
const KEvPtrExit&=&40A

rem Pointer event types
const KEvPtrPenDown&=0
const KEvPtrPenUp&=1
const KEvPtrButton1Down&=KEvPtrPenDown&
const KEvPtrButton1Up&=KEvPtrPenUp&
const KEvPtrButton2Down&=2
const KEvPtrButton2Up&=3
const KEvPtrButton3Down&=4
const KEvPtrButton3Up&=5
const KEvPtrDrag&=6
const KEvPtrMove&=7
const KEvPtrButtonRepeat&=8
const KEvPtrSwitchOn&=9
```

```
rem For PointerFilter
const KPointerFilterEnterExit%=$1
const KPointerFilterMove%=$2
const KPointerFilterDrag%=$4

rem code page 1252 ellipsis (“windows latin 1”)
const KScreenEllipsis%=133
const KScreenLineFeed%=10

rem For gCLOCK
const KClockLocaleConformant%=6
const KClockSystemSetting%=KClockLocaleConformant%
const KClockAnalog%=7
const KClockDigital%=8
const KClockLargeAnalog%=9
rem gClock 10 no longer supported (use slightly changed gCLOCK 11)
const KClockFormattedDigital%=11

rem For gFONT (these may change before release)

const KFontArialBold8&=       268435951
const KFontArialBold11&=      268435952
const KFontArialBold13&=      268435953
const KFontArialNormal8&=     268435954
const KFontArialNormal11&=    268435955
const KFontArialNormal13&=    268435956
const KFontArialNormal15&=    268435957
const KFontArialNormal18&=    268435958
const KFontArialNormal22&=    268435959
const KFontArialNormal27&=    268435960
const KFontArialNormal32&=    268435961

const KFontTimesBold8&=       268435962
const KFontTimesBold11&=      268435963
const KFontTimesBold13&=      268435964
const KFontTimesNormal8&=     268435965
const KFontTimesNormal11&=    268435966
const KFontTimesNormal13&=    268435967
const KFontTimesNormal15&=    268435968
const KFontTimesNormal18&=    268435969
const KFontTimesNormal22&=    268435970
const KFontTimesNormal27&=    268435971
const KFontTimesNormal32&=    268435972

const KFontCourierBold8&=     268436062
const KFontCourierBold11&=    268436063
const KFontCourierBold13&=    268436064
const KFontCourierNormal8&=   268436065
const KFontCourierNormal11&=  268436066
const KFontCourierNormal13&=  268436067
const KFontCourierNormal15&=  268436068
const KFontCourierNormal18&=  268436069
const KFontCourierNormal22&=  268436070
const KFontCourierNormal27&=  268436071
const KFontCourierNormal32&=  268436072
```

```
const KFontCalc13n&=    268435493
const KFontCalc18n&=   268435494
const KFontCalc24n&=   268435495

const KFontMon18n&=    268435497
const KFontMon18b&=    268435498
const KFontMon9n&=     268435499
const KFontMon9b&=     268435500

const KFontTiny1&=     268435501
const KFontTiny2&=     268435502
const KFontTiny3&=     268435503
const KFontTiny4&=     268435504

const KFontEiksym15&=  268435661

const KFontSquashed&=  268435701
const KFontDigital35&= 268435752

rem For IOOPEN
rem Mode category 1
const KIoOpenModeOpen%=$0000
const KIoOpenModeCreate%=$0001
const KIoOpenModeReplace%=$0002
const KIoOpenModeAppend%=$0003
const KIoOpenModeUnique%=$0004

rem Mode category 2
const KIoOpenFormatBinary%=$0000
const KIoOpenFormatText%=$0020

rem Mode category 3
const KIoOpenAccessUpdate%=$0100
const KIoOpenAccessRandom%=$0200
const KIoOpenAccessShare%=$0400

rem Language code for CAPTION
const KLangEnglish%=1
const KLangFrench%=2
const KLangGerman%=3
const KLangSpanish%=4
const KLangItalian%=5
const KLangSwedish%=6
const KLangDanish%=7
const KLangNorwegian%=8
const KLangFinnish%=9
const KLangAmerican%=10
const KLangSwissFrench%=11
const KLangSwissGerman%=12
const KLangPortuguese%=13
const KLangTurkish%=14
const KLangIcelandic%=15
const KLangRussian%=16
const KLangHungarian%=17
const KLangDutch%=18
const KLangBelgianFlemish%=19
```

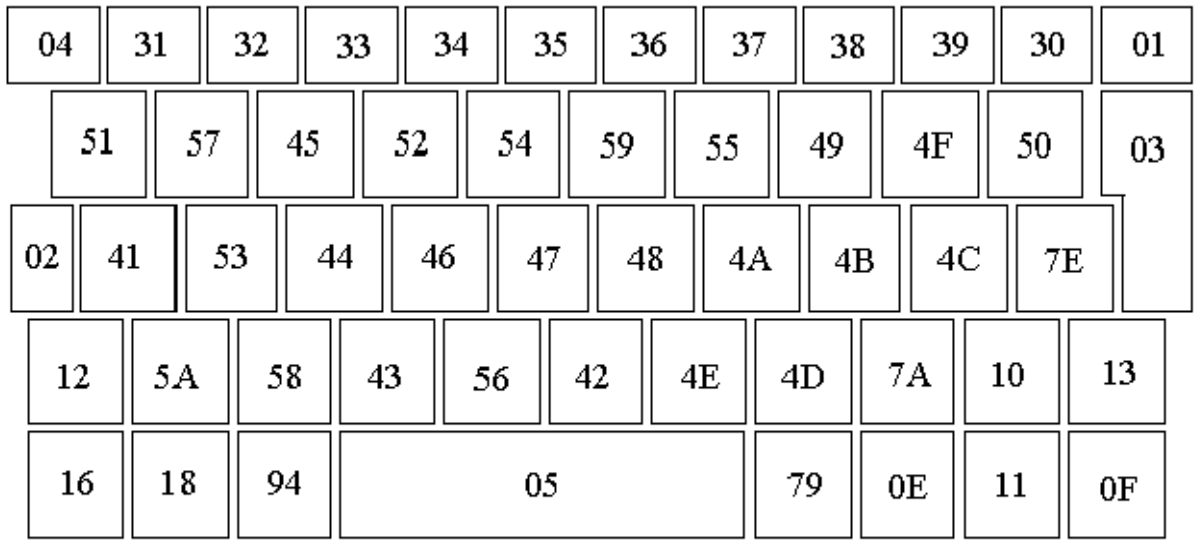
```
const KLangAustralian%=20
const KLangBelgianFrench%=21
const KLangAustrian%=22
const KLangNewZealand%=23
const KLangInternationalFrench%=24

rem End of Const.oph
```

APPENDIX B

SCANCODES FOR THE SERIES 5 KEYBOARD

All values are given in hexadecimal.



APPENDIX C

SQL SPECIFICATION

NOTE ON SYNTAX

The use of square brackets [ ] indicates that something is optional, while a vertical line | indicates a choice shold be made between mutually exclusive options. Words in heavy mono-spaced type (e.g. **SELECT**) should be typed in literally .

SQL keywords are case insensitive.

SELECT STATEMENT

select-statement :

> **SELECT** *select-list*
> **FROM** *table-name*
> [**WHERE** *search-condition*]
> [**ORDER BY** *sort-order*]

Use a *select-statement* to specify what data should be present in the Rowset, and how to present it.

SELECTING COLUMNS

select-list :
> *****
> *column-name-comma-list*

Specify ***** to request that all columns for the table be returned in the Rowset, in an undefined order; otherwise a comma separated list specifies which columns to return, and the order that the columns appear in the Rowset.

NAMES

table-name

column-name

The table-name should be a table which exists in the database. Column names should refer to columns which exist in the specified table.

SEARCH CONDITION

search-condition :
> *boolean-term* [**OR** *search-condition*]

boolean-term :
> *boolean-factor* [**AND** *boolean-term*]

boolean-factor :
> [**NOT**] *boolean-primary*

boolean-primary :
> *predicate*
> **(** *search-condition* **)**

This specifies a condition which a row must meet to be present in the generated Rowset. A trivial search condition is just a single predicate, more complex search conditions are constructed by combining predicates using the keywords **AND**, **OR** and **NOT**, and using parantheses to override the standard precedence of these operators. Without brackets, the order of precedence is **NOT**, **AND** then **OR**. e.g.

```
a=1 or not b=2 and c=3
```

is equivalent to

```
(a=1 or ((not b=2) and c=3))
```

PREDICATES

predicate :
comparison-predicate
like-predicate
null-predicate

These are the building blocks of the search condition. Each predicate tests one condition of a column in the selected table.

COMPARISON PREDICATE

comparison-predicate :
column-name comparison-operator literal

comparison-operator :
< / > / <= / >= / = / <>

Compare a column value with a supplied literal value. Numeric columns (including bit columns) are compared numerically, text columns are compared lexically and date columns are compared historically. Binary columns cannot be compared. The literal must be of the same type (numeric, string, date) as the column.

LITERALS

literal :
string-literal
numeric-literal
date-literal

A *string-literal* is a character string enclosed in single quote characters ‘. To include a single literal quote character ‘ in a *string-literal*, use two literal quote characters ‘’.

A *numeric-literal* is any sequence of characters which can be interpreted as a valid decimal integral or floating point number.

A *date-literal* is a character string enclosed by the # character, which can be interpreted as a valid date..

LIKE PREDICATE

like-predicate :
    column-name **[NOT] LIKE** *pattern-value*

pattern-value :
    string-literal

Test whether or not a text column matches a pattern string. The wildcard characters used in the *pattern-value* are not standard SQL, instead the EPOC32 wildcard characters are used: ? for matching any single character and * for matching zero or more characters.

NULL PREDICATE

null-predicate :
    column-name **IS [NOT] NULL**

Test whether or not a column is Null. This predicate can be applied to all column types.

SPECIFYING A SORT ORDER

sort-order :
    sort-specification-comma-list

sort-specification :
    column-name [**ASC | DESC**]

Without an **ORDER BY** clause in the select statement the order that rows are presented is undefined. The columns specified in the sort-order can be ordered in ascending (the default) or descending order, and should appear in the sort-order in decreasing order of precedence. e.g.

```
surname, first_name
```

will order the rows by the column `surname`, and any rows with identical surnames will then be ordered by the column `first_name`.