OPL

APPENDICES

CONTENTS

APPENDIX A

SUMMARY FOR EXPERIENCED OPL USERS

OPL has evolved from the Psion Organiser II through the MC and HC computers to the Series 3 and the Series 3a and then to the Series 3c and Siena and to the Series 5. This appendix gives a summary of the changes made from the Series 3a upwards. *For more details of the following topics and keywords, look them up in the 'Alphabetical listing' section of the 'Glossary.pdf' document.*

Bear in mind that some OPL keywords return or allow different values according to screen size and keyboard layout.

USING OPL ON THE SERIES 3A, SERIES 3C AND SIENA

OPL on the Series 3c is similar to that on the Series 3a except for some differences with serial cables.

Again the Siena is similar to these two, except for the size of its screen being around half that of the other two and also the fact that it does not have an SSD.

USING OPL ON THE SERIES 5

The principal design requirements of OPL for the Series 5 were:

- Compatibility with earlier versions of 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.

The major difference between this and other versions of OPL is that 32-bit rather than 16-bit addressing is used. This means that the arguments and return values of quite a number of keywords have changed from being integers to long integers.

Also graphics, menus, dialogs and database handling especially have been improved to take advantage of the abilities of EPOC32.

Below the removed, new and changed features are listed. For this appendix to provide detailed information would represent a repetition of much of what has been described in the main chapters of this manual and you should refer to these for full details (especially useful is the 'Alphabetic Listing' which includes full details of all keywords for both the Series 5 and the Series 3c).

The following keywords, available on earlier Psion machines (Series 3a, Series 3c and Siena), are no longer available on the Series 5:

- RECSIZE and COMPRESS. COMPACT replaces COMPRESS.
- gDRAWOBJECT.
- **gINFO. This is replaced by gINFO32.**
- STATUSWIN, STATWININFO, DIAMINIT and DIAMPOS. The Series 5 has no status windows. Toolbars are used instead See the 'Toolbar Usage' section in the 'Friendlier Interaction' chapter.
- TYPE, PATH and EXT. These were provided on earlier machines for use by the System screen only. The Series 5 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. (A new command FLAGS has a similar function to TYPE.) See the 'Advanced Topics' chapter.
- CMD$(4) and CMD$(5).
- SETNAME. The new SETDOC, serves a similar purpose, and should be called before saving your main document.
- Named Calculator memories and M0,...,M9. The Series 5 Calculator does not use OPL to evaluate expressions.
- CACHE, CACHETIDY, CACHEHDR and CACHEREC. On the Series 5 procedures are automatically cached.

- CREATESPRITE, APPENDSPRITE, CHANGESPRITE, DRAWSPRITE, POSSPRITE and CLOSESPRITE. Superior sprite-handling OPX functions are provided. See the ‘Sprite and Bitmap OPX’ section in the ‘Using OPXs on the Series 5’ chapter.
- OS and CALL. The Series 5 provides extensibility using special OPL DLLs. See the ‘Using OPXs on the Series 5’ chapter.
- USR and USR$.
- ODBINFO. This is implementation specific.
- LOADLIB, LINKLIB, UNLOADLIB, FINDLIB, GETLIBH, NEWOBJ, NEWOBJH, SEND, ENTERSEND and ENTERSEND0. OPXs are now used for calling language extensions and creating object instances. See the ‘Using OPXs on the Series 5’ chapter.

The following keywords have been added to OPL on the Series 5:

- DECLARE EXTERNAL, EXTERNAL, INCLUDE and CONST allow the use of header files which include the definition of constants and procedure prototypes.
- mCASC and mPOPUP provide new menu features.
- dCHECKBOX and dEDITMULTI provide new features for dialogs.
- DAYSTODATE allows easy conversion of “days since 1/1/1990” to a date.
- gCOLOR, gCIRCLE, gELLIPSE and gSETPENWIDTH provide new graphics functionality.
- gINFO32 replaces gINFO.
- SETFLAGS and CLEARFLAGS allow for Series 3a/Series 3c/Siena compatibility
- IOWAITSTAT32.

Database commands:

- DELETE allows deletion of a table.
- INSERT, MODIFY, PUT and CANCEL allow the building up and editing of databases.
- BOOKMARK, KILLMARK, GOTOMARK support the use of bookmarks in databases.
- BEGINTRANS, COMMITTRANS, INTRANS and ROLLBACK support transactions in databases.
- COMPACT replaces COMPRESS.

OPL applications:

- CAPTION and FLAGS allow definition of OPL applications; FLAGS is similar to TYPE.
- SETDOC and GETDOC$ allow files to be created as documents.
- GETEVENT32, GETEVENTA32 and POINTERFILTER provide increased support for handling of events, including pointer (pen) events.

Two other major additions have also been made to OPL. These are:

- Support for Toolbars (to replace status windows). See the ‘Toolbar Usage’ section in the ‘Friendlier Interaction’ chapter.
- Support for language extensions in separate EPOC32 DLLs called OPXs. See the ‘Using OPXs on the Series 5’ chapter.

The following keywords have undergone some changes, although many of these are compatible with earlier versions of OPL:

- ADDR, ALLOC, ADJUSTALLOC, REALLOC, LENALLOC and FREEALLOC
- APP and ICON
- CMD$(3)
- GETEVENT
- BUSY
- OFF
- SCREENINFO
- dBUTTONS, dCHOICE, dFILE, dINIT, dTEXT, dTIME and dXINPUT
- mCARD
- CLOSE, COUNT, CREATE, OPEN, POS and POSITION
- CURSOR, DEFAULTWIN, gBORDER, gBUTTON, gCLOCK, gCREATE, gCREATEBIT, gFONT, gGREY, gLINETO, gLINEBY, gLOADBIT, gSAVEBIT, gLOADFONT, gUNLOADFONT, gPEEKLINE and gXBORDER.

APPENDIX B

OPERATORS AND LOGICAL EXPRESSIONS

OPERATORS

These operators are available in OPL:

ARITHMETIC OPERATORS

`+` add

`-` subtract

`*` multiply

`/` divide

`**` raise to a power

`-` unary minus (in negative numbers for example, `-10`)

`%` percent

COMPARISON OPERATORS

`>` greater than

`>=` greater than or equal to

`<` less than

`<=` less than or equal to

`=` equal to

`<>` not equal to

LOGICAL AND BITWISE OPERATORS

`AND`

`OR`

`NOT`

THE % OPERATOR

The percentage operator can be used in expressions like this:

Expression	*Meaning*	*Result*
`60+5%`	60 plus 5% of 60	63
`60-5%`	60 minus 5% of 60	57
`60*5%`	5% of 60	3
`60/5%`	of what number is 60 5%?	1200

It can also be used like this:

`105>5%`	what number, when increased by 5%, becomes 105?	100
`105<5%`	how much of 105 is a 5% increase?	5

Examples:

To add 15% tax to 345:

`345+15%` Result = 396.75

To find out what the price was before tax:

`396.75>15%` Result = 345

To find out how much of a total price is tax:

`396.75<15%` Result = 51.75

5 OTHER OPERATORS

On the Series 5, the System OPX provides two addtional operators:

`MOD&:(x&,y&)` `x&` modulo `y&` (the remainder of x& divided by y&)

`XOR&:(x&,y&)` exclusive OR of `x&` and `y&`

PRECEDENCE

Highest: `**`

`-` (unary minus)

`NOT`

`*` `/`

`+` `-` (subtraction)

`=` `>` `<` `<>` `>=` `<=`

Lowest: `AND` `OR`

So `7+3*4` returns 19 (3 is first multiplied by 4, and 7 is added to the result) not 40 (4 times 10).

WHEN THERE IS EQUAL PRECEDENCE

In an expression where all operators have equal precedence, they are evaluated from left to right (with the exception of powers). For example, in `a+b-c`, `a` is added to `b` and then `c` is subtracted from the result.

Powers are evaluated from right to left for example, in `a%**b%**c%`, `b%` will first be raised to the power of `c%` and then `a%` will be raised to the power of the result.

CHANGING PRECEDENCE WITH BRACKETS

The result of an expression such as `a+b+c` is the same whether you first add `a` to `b`, or `b` to `c`. But how is `a+b*c/d` evaluated? You may want to use brackets to either:

- Make it obvious what the order of calculation is

or

- Change the order of calculation.

By default, `a+b*c/d` is evaluated in the order: `b` multiplied by `c`, then divided by `d`, then added to `a`. To perform the addition and the division before the multiplication, use brackets: `(a+b)*(c/d)`. When in doubt, simply use brackets.

PRECEDENCE OF INTEGER AND FLOATING-POINT VALUES

You are free to mix floating-point and integer values in expressions, but be aware how OPL handles the mix:

- In each part of the calculation, OPL uses the simplest arithmetic possible. Two integers will use integer arithmetic, and this can give unexpected results: **`7/2` gives the integer `3`**. Otherwise floating-point arithmetic is used (`7.0` is a floating-point number, so `7.0/2` gives the floating-point number `3.5`).
- Finally, the evaluated result of the right-hand side of an expression is automatically converted to the same type as the variable to which it is assigned.

For example, your procedure might include the expression `a%=b%+c` This is handled like this: `b%` is converted to floating-point and added to `c`. The resulting floating-point value is then automatically converted to an integer in order to be assigned to the integer variable `a%`.

Such conversions may produce odd results for example `a%=3.0*(7/2)` makes `a%=9`, but `a%=3.0*(7.0/2)` makes `a%=10`. OPL does not report this as an error, so it's up to you to ensure that it doesn't happen unless you want it to.

TYPE CONVERSIONS AND ROUNDING DOWN

There are three numeric types floating-point, integer and long integer. You can assign any of these types to any other. The value on the right-hand side will be automatically converted to the type of the variable on the left-hand side. For example:

- If you assign an integer value to a floating-point variable, there are no problems.
- If you assign a floating-point value to an integer variable, the value is converted to an integer, always rounded **towards zero** for example, if you declare `LOCAL c%` and then say `c%=3.75`, the value `3.75` is converted to the value `3`.

Rounding down towards zero can sometimes cause unusual results. For example, `a%=2.9` would give `a%` the value 2, and `a%=-2.3` would give `a%` the value `-2`.

When you run a module, if the left-hand side of an assignment has a narrower range than the right-hand side, you may get an error (for example, if you had `x%=a&` where `a&` had the value `320000`).

To control how floating-point numbers are rounded when converted, use the INT function.

LOGICAL EXPRESSIONS

The comparison operators and logical operators are based on the idea that a certain situation can be evaluated as either true or false. For example, if `a%=6` and `b%=8`, `a%>b%` would be 'False'.

These operators are useful for setting up alternative paths in your procedures. For example you could say:

```
IF salary<expenses
  doBad:
ELSE
  doGood:
ENDIF
```

You can also make use of the fact that the result of these logical expressions is represented by an integer:

- 'True' is represented by the integer -1
- 'False' is represented by the integer 0 (zero).

operator	*example*	*result returned*	*return value*
<	a<b	True if a less than b False if a greater than or equal to b	-1 0
>	a>b	True if a greater than b False if a less than or equal to b	-1 0
<=	a<=b	True if a less than or equal to b False if a greater than b	-1 0
=	a>=b	True if a greater than or equal to b False if a less than b	-1 0
<>	a<>b	True if a not equal to b False if a equal to b	-1 0
=	a=b	True if a equal to b False if a not equal to b	-1 0

These integers can be assigned to a variable or displayed on the screen to tell you whether a particular condition is true or false, or used in an IF statement.

For example, in a procedure you might arrive at two sub-totals, a and b. You want to find out which is the greater. So use the statement, `PRINT a>b`. If zero is displayed, a and b are equal or b is the larger number; if -1 is displayed, a>b is true a is the larger.

LOGICAL AND BITWISE OPERATORS

The operators AND, OR and NOT have different effects depending on whether they are used with floating-point numbers or integers:

WHEN USED WITH FLOATING-POINT NUMBERS

AND, OR and NOT are *logical operators*, and have the following effects:

example	*result*	*integer returned*
`a AND b`	True if both a and b are non-zero False if either a or b are zero	-1 0
`a OR b`	True if either a or b is non-zero False if both a and b are zero	-1 0
`NOT a`	True if a is zero False if a is non-zero	-1 0

WHEN USED WITH INTEGER OR LONG INTEGER VALUES

AND, OR and NOT are *bitwise operators*.

The way OPL holds integer numbers internally is as a binary code - 16-bit for integers, 32-bit for long integers. Bitwise means that an operation is performed on individual bits. A bit is *set* if it has the value 1, and *clear* if it has the value 0. Long integer values with AND, OR and NOT behave the same as integer values.

AND

Sets the result bit if both input bits are set, otherwise clears the result bit.

For example, the statement `PRINT 12 AND 10` displays 8. To understand this, write 12 and 10 in binary:

```
12   0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
10   0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

AND acts on each pair of bits. Thus, working from left to right discounting the first 12 bits (since `0 AND 0` gives 0):

```
1 AND 1   →   1
1 AND 0   →   0
0 AND 1   →   0
0 AND 0   →   0
```

The result is therefore the binary number 1000, or 8.

OR

Sets the result bit if either input bit is set, otherwise clears the result bit.

What result would the statement `PRINT 12 OR 10` give? Again, write down the numbers in binary and apply the operator to each pair of digits:

```
1 OR 1    →   1
1 OR 0    →   1
0 OR 1    →   1
0 OR 0    →   0
```

The result is the binary number 1110, or 14 in decimal.

NOT

*Sets the result bit if the input bit is **not** set, otherwise clears the result bit.*

NOT works on only one number. It returns the *one's complement*, i.e. it replaces 0s with 1s and 1s with 0s.

So since 7 is 0000000000000111, `NOT 7` is 1111111111111000. This is the binary representation of -8.

A quick way of calculating NOT for integers is to add 1 to the original number and reverse its sign. So `NOT 23` is -24, `NOT 0` is -1 and `NOT -1` is 0.

APPENDIX C

SERIAL/PARALLEL PORTS AND PRINTING

OPL

You can use LPRINT in an OPL program to send information (for printing or otherwise) to any of these devices:

- ❸ A parallel port, PAR:A
- A serial port, TTY:A
- A file on the Psion
- ❸ A file on an attached computer, e.g. on a PC or on an Apple Macintosh:

❸ OPL does not provide access to the advanced page formatting and font control features of the Series 3c.

❺ OPL provides more advanced formatting features and printing control using Printer OPX. See the 'Using OPXs on the Series 5' chapter for more details.

You can also read information from the serial port.

USING THE PARALLEL PORT ON THE SERIES 3C AND SIENA

In your OPL program, set up the port with the statement `LOPEN "PAR:A"`.

Provided the port is not already in use, the connection is now ready. LPRINT will send information down the parallel 3 Link lead for example, to an attached printer.

EXAMPLE

```
PROC prints:
   OPEN "clients",A,a$
   LOPEN "PAR:A"
   PRINT "Printing..."
   DO
     IF LEN(A.a$)
          LPRINT A.a$
     ENDIF
     NEXT
   UNTIL EOF
   LPRINT CHR$(12); :LCLOSE
   PRINT "Finished" :GET
ENDP
```

USING THE SERIAL PORT

In your OPL program, set up the port with the statement `LOPEN "TTY:A"`.

Now LPRINT should send information down the serial link lead for example, to an attached printer. If it does not, the serial port settings are not correct.

SERIAL PORT SETTINGS

`LOPEN "TTY:A"` opens the serial port with the following default characteristics:

9600 baud

no parity

8 data bits

1 stop bit

RTS handshaking.

- If your printer (or other device) **does** match these characteristics, the LOPEN statement sets the port up correctly, and subsequent LPRINT statements will print there successfully.
- If your printer **does not** match these characteristics, you must use a procedure like the one listed below to change the characteristics of the serial port, before LPRINTs will print successfully to your printer.

Printers very often use DSR (DSR/DTR) handshaking, and you may need to set the port to use this.

SETTING THE SERIAL PORT CHARACTERISTICS

Calling the procedure

The `rsset:` procedure listed below provides a convenient way to set up the serial port.

Each time you use an `LOPEN "TTY:"` statement, follow it with a call to the `rsset:` procedure. Otherwise the LOPEN will use the default characteristics.

Passing values to the procedure

Pass the procedure the values for the five port characteristics, like this:

```
rsset:(baud%,parity%,data%,stop%,hand%,&0)
```

The final parameter, which should be `&0` here, is only used when reading from the port.

To find the value you need for each characteristic, use the tables below. You **must** give values to all five parameters, in the correct order.

Baud	50	75	110	134	150	300	600	1200	1800	2000	2400	3600	4800	7200	9600	19200
value	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Parity	NONE	EVEN	ODD
value	0	1	2

Data bits	5, 6, 7 or 8
Stop bits	2 or 1

Handshaking	ALL	NONE	XON	RTS	XON+RTS	DSR	XON+DSR	RTS+DSR
value	11	4	7	0	3	12	15	8

THE `rsset:` PROCEDURE:

```
PROC rsset:(baud%,parity%,data%,stop%,hand%,term&)
   LOCAL frame%,srchar%(6),dummy%,err%
   frame%=data%-5
   IF stop%=2 :frame%=frame% OR 16 :ENDIF
   IF parity% :frame%=frame% OR 32 :ENDIF
   srchar%(1)=baud% OR (baud%*256)
   srchar%(2)=frame% OR (parity%*256)
   srchar%(3)=(hand% AND 255) OR $1100
   srchar%(4)=$13
   POKEL ADDR(srchar%(5)),term&
   err%=IOW(-1,7,srchar%(1),dummy%)
   IF err% :RAISE err% :ENDIF
ENDP
```

Take care to type this program in exactly as it appears here.

EXAMPLE OF CALLING THE PROCEDURE

```
PROC test:
   PRINT “Testing port settings”
   LOPEN “TTY:A”
   LOADM “rsset”
   rsset:(8,0,8,1,0,&0)
   LPRINT “Port OK” :LPRINT
   PRINT “Finished” :GET
   LCLOSE
ENDP
```

`rsset:(8,0,8,1,0,&0)` sets 1200 Baud, no parity, 8 data bits, 1 stop bit, and RTS/CTS handshaking.

ADVANCED USE

The section of the `rsset:` procedure which actually sets the port is this:

```
srchar%(1)=baud% OR (baud%*256)
srchar%(2)=frame% OR (parity%*256)
srchar%(3)=(hand% AND 255) OR $1100
srchar%(4)=$13
POKEL ADDR(srchar%(5)),term&
err%=IOW(-1,7,srchar%(1),dummy%)
IF err% :RAISE err% :ENDIF
```

The elements of the array `srchar%` contain the values specifying the port characteristics. If you want to write a shorter procedure, you could work out what these values need to be for a particular setup you want, assign these values to the elements of the array, and then use the IOW function (followed by the error check) **exactly** as above.

READING FROM THE SERIAL PORT

If you need to read from the serial port, you must also pass a parameter specifying terminating mask for the read function. If `term&` is not supplied, the read operation terminates only after reading exactly the number of bytes requested. In practice, however, you may not know exactly how many bytes to expect and you would therefore request a large maximum number of bytes. If the sender sends less than this number of bytes altogether, the read will never complete.

The extra parameter, `term&` , allows you to specify that one or more characters should be treated as terminating characters. The terminating character itself is read into your buffer too, allowing your program to act differently depending on its value.

The 32 bits of `term&` each represent the corresponding ASCII character that should terminate the read. This allows any of the ASCII characters 1 to 31 to terminate the read.

For example, to terminate the read when Control-Z (i.e. ASCII 26) is received, set bit 26 of `term&`. To terminate on Control-Z or `<CR>` or `<LF>` which allows text to be read a line at a time or until end of file set the bits 26, 10 and 13. In binary, this is:

0000 0100 0000 0000 0010 0100 0000 0000

Converting to a long integer gives &04002400 and this is the value to be passed in `term&` for this case.

Clearly `term&` cannot be used for binary data which may include a terminating character by chance. You can sometimes get around this problem by using `term&` and having the sender transmit a leading non-binary header specifying the exact number of full-binary data following. You could then reset the serial characteristics not to use `term&` , read the binary data, and so forth.

EXAMPLE READING FROM SERIAL PORT

This example assumes that each line sent has maximum length 255 characters and is terminated by a `<CR>` and that Control-Z signals the end of all the data.

```
PROC testread:
  LOCAL ret%,pbuf&,pbuf1&,buf$(255),end%,len%
  PRINT "Test reading from serial port"
  LOPEN "TTY:A"
  LOADM "rsset"                  REM receive at 2400 without h/shake
  rsset:(11,0,8,1,0,&04002000)   REM Control-Z or CR
  pbuf&=ADDR(buf$)               REM could be pbuf% on Series 3c
  DO                    REM read max 255 bytes, after leading count byte
    len%=255
    pbuf1&=pbuf&+1
    ret%=IOW(-1,1,#pbuf1&,len%)
    POKEB pbuf&,len%             REM len% = length actually read
    REM including terminator char
    end%=LOC(buf$,CHR$(26))      REM non-zero for Control-Z
    IF ret%<0 and ret%<>-43
         BEEP 3,500
         PRINT
         PRINT "Serial read error: ";ERR$(ret%)
    ENDIF
    IF ret%<>-43                 REM if received with terminator
         POKEB pbuf&,len%-1      REM remove terminator
         PRINT buf$              REM echo with CRLF
```

```
    ELSE
        PRINT buf$;             REM echo without CRLF
    ENDIF
  UNTIL end%
  PRINT "End of session" :PAUSE -30 :KEY
ENDP
```

Note that passing -1 as the first argument to I/O keywords means that the LOPEN handle is to be used. Also, OPL strings have a leading byte giving the length of the rest of the string, so the data is read beyond this byte. The byte is then poked to the length which was read.

PRINTING TO A FILE

PRINTING TO A FILE ON THE PSION

In your OPL program, specify the destination file with an LOPEN statement like this:

❺ `LOPEN "D:\PRINT\MEMO"`

❸ `LOPEN "B:\PRINT\MEMO.TXT"`

❸ PRINTING TO A FILE ON A PC OR APPLE MACINTOSH

As if you were going to transfer a file:

- Physically connect the Psion and the other computer.
- Select the 'Remote link' option in the System screen and press Enter.
- Run the server program (supplied with 3 Link if you are using a Series 3c or Siena, or with the Series 5 itself) on the other computer.

In your OPL program, specify the destination file with an LOPEN statement.

For example, to a PC:

```
LOPEN "REM::C:\BACKUP\PRINTOUT\MEMO.TXT"
```

Any subsequent LPRINT would go to the file `MEMO.TXT` in the directory `\BACKUP\PRINTOUT` on the PC's drive `C:`.

With a Macintosh, you might use a file specification like this:

```
LOPEN "REM::HD40:MY BACKUP:PRINTED:MEMO5"
```

An LPRINT would now go to the file `MEMO5` in the `PRINTED` folder, itself in the `MY BACKUP` folder on the hard drive `HD40`. Note that colons are used to separate the various parts of the file specification.

APPENDIX D

CHARACTER CODES

THE CHARACTER SET FOR THE SERIES 5

The following are character code descriptions for the 8-bit character set used in the English version of EPOC32. The definition is intended to be synonymous with code page 1252 (also used by Microsoft for Windows systems).

Those items in the 'character' column which are given in italics are descriptions. Particularly note that the code for *backspace* will be the code returned when you press Del, that for *carriage return* will be returned when you press Enter and that for *escape* when you press Esc.

hex code	*dec. code*	*character*	*hex code*	*dec. code*	*character*
00	000	*null*	01	001	*start of heading*
02	002	*start of text*	03	003	*end of text*
04	004	*end of transmission*	05	005	*enquiry*
06	006	*acknowledge*	07	007	*bell*
08	008	*backspace*	09	009	*horizontal tabulation*
0A	010	*line feed*	0B	011	*vertical tabulation*
0C	012	*form feed*	0D	013	*carriage return*
0E	014	*shift out*	0F	015	*shift in*
10	016	*data link escape*	11	017	*device control one*
12	018	*device control two*	13	019	*device control three*
14	020	*device control four*	15	021	*negative acknowledge*
16	022	*synchronous idle*	17	023	*end of transmission block*
18	024	*cancel*	19	025	*end of medium*
1A	026	*substitute*	1B	027	*escape*
1C	028	*file separator*	1D	029	*group separator*
1E	030	*record separator*	1F	031	*unit separator*
20	032	(*space*)	21	033	!

hex code	*dec. code*	*character*	*hex code*	*dec. code*	*character*	*hex code*	*dec code*	*character*
22	034	"	23	035	#	24	036	$
25	037	%	26	038	&	27	039	'
28	040	(	9	041	)	2A	042	*
2B	043	+	2C	044	,	2D	045	-
2E	046	.	2F	047	/	30	048	0
31	049	1	32	050	2	33	051	3
34	052	4	35	053	5	36	054	6

37	055	7	38	056	8	39	057	9
3A	058	:	3B	059	;	3C	060	<
3D	061	=	3E	062	>	3F	063	?
40	064	@	41	065	A	42	066	B
43	067	C	44	068	D	45	069	E
46	070	F	47	071	G	48	072	H
49	073	I	4A	074	J	4B	075	K
4C	076	L	4D	077	M	4E	078	N
4F	079	O	50	080	P	51	081	Q
52	082	R	53	083	S	54	084	T
55	085	U	56	086	V	57	087	W
58	088	X	59	089	Y	5A	090	Z
5B	091	[	5C	092	/	5D	093	]
5E	094	^	5F	095	_	60	096	‘
61	097	a	62	098	b	63	099	c
64	100	d	65	101	e	66	102	f
67	103	g	68	104	h	69	105	i
6A	106	j	6B	107	k	6C	108	l
6D	109	m	6E	110	n	6F	111	o
70	112	p	71	113	q	72	114	r
73	115	s	74	116	t	75	117	u
76	118	v	77	119	w	78	120	x
79	121	y	7A	122	z	7B	123	{
7C	124	|	7D	125	}	7E	126	~
7F	127	*delete*	80	128	*not used*	81	129	*not used*
82	130	,	83	131	*f*	84	132	„
85	133	…	86	134	†	87	135	‡
88	136	ˆ	89	137	‰	8A	138	Š
8B	139	‹	8C	140	Œ	8D	141	*not used*
8E	142	*not used*	8F	143	*not used*	90	144	*not used*
91	145	‘	92	146	’	93	147	“
94	148	”	95	149	•	96	150	–
97	151	—	98	152	˜	99	153	™
9A	154	š	9B	155	›	9C	156	œ

9D	157	*not used*	9E	158	*not used*	9F	159	Ÿ
A0	160	*no-break space*	A1	161	¡	A2	162	¢
A3	163	£	A4	164	¤	A5	165	¥
A6	166	¦	A7	167	§	A8	168	¨
A9	169	©	AA	170	ª	AB	171	«
AC	172	¬	AD	173	-	AE	174	®
AF	175	¯	B0	176	°	B1	177	±
B2	178	²	B3	179	³	B4	180	´
B5	181	µ	B6	182	¶	B7	183	·
B8	184	¸	B9	185	¹	BA	186	º
BB	187	»	BC	188	¼	BD	189	½
BE	190	¾	BF	191	¿	C0	192	À
C1	193	Á	C2	194	Â	C3	195	Ã
C4	196	Ä	C5	197	Å	C6	198	Æ
C7	199	Ç	C8	200	È	C9	201	É
CA	202	Ê	CB	203	Ë	CC	204	Ì
CD	205	Í	CE	206	Î	CF	207	Ï
D0	208	Đ	D1	209	Ñ	D2	210	Ò
D3	211	Ó	D4	212	Ô	D5	213	Õ
D6	214	Ö	D7	215	×	D8	216	Ø
D9	217	Ù	DA	218	Ú	DB	219	Û
DC	220	Ü	DD	221	Ý	DE	222	Þ
DF	223	ß	E0	224	à	E1	225	á
E2	226	â	E3	227	ã	E4	228	ä
E5	229	å	E6	230	æ	E7	231	ç
E8	232	è	E9	233	é	EA	234	ê
EB	235	ë	EC	236	ì	ED	237	í
EE	238	î	EF	239	ï	F0	240	ð
F1	241	ñ	F2	242	ò	F3	243	ó
F4	244	ô	F5	245	õ	F6	246	ö
F7	247	÷	F8	248	ø	F9	249	ù
FA	250	ú	FB	251	û	FC	252	ü
FD	253	ý	FE	254	þ	FF	255	ÿ

ENTERING CHARACTERS AT THE KEYBOARD

Any of the characters in the character set can be entered directly from the keyboard:

1. Look up the **decimal** code of the character you want from the preceding table.
2. Hold down the Ctrl key and type the three-digit code, then release the control key.

Make sure that you include any preceding zeros to make the code three digits long - for example use Ctrl+096 to enter a single left quote, ‘.

MODIFICATION OF CHARACTER CODES

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. For example, pressing Ctrl+J returns 10 which is (`%j AND (NOT $60)`).

OTHER SPECIAL KEYS

The keycodes (returned by GETEVENT32, etc) for these special keys are as follows:

key	*hex*	*decimal*	*key*	*hex*	*decimal*
Home	1002	4098	End	1003	4099
PgUp	1004	4100	PgDn	1005	4101
◂	1007	4103	▸	1008	4104
▲	1009	4105	▼	100A	4106
Menu	1036	4150			

GET and KEY, however, return the same values as on the Series 3c (see below).

CHARACTER CODES ON THE SERIES 3C AND SIENA

To find out a character’s character code either look up the character in the table given in the back of your User Guide, or press the Calc button and type the % sign followed by the character for example %P returns 80. Characters with codes from 0 to 127 are the same as in the ASCII character set. Codes 128 to 255 are compatible with the IBM code page 850.

Codes from 256 upwards are for other Series 3c keys see the list below.

CHARACTER CODES OF SPECIAL KEYS

The GET and KEY functions return the character code of the key that was pressed. Some of the keys are not in the character set. They return these numbers:

Esc	27	Tab	9
Delete	8	Enter	13
↑	256	↓	257
→	258	←	259

Pg Up	260	Pg Dn	261
Home	262	End	263
Menu	290	Help	291
	292		

The Psion key adds 512 to the value of the key pressed. For example, Psion-a is 609 (512+97), and Psion-Help (Dial) is 803 (512+291).

SPECIAL CHARACTER CODES WITH PRINT

These values can be used with PRINT and CHR$():

7	beep
8	backspace
9	tab
10	line feed
12	form feed (clear screen)
13	carriage return (cursor to left of window)

For example, `PRINT CHR$(8)` moves the cursor backwards, one character to the left.

KEYBOARD MODIFIERS

Keyboard modifier values are the same for all machines, except that the Psion key (modifier value 8) does not exist on the Series 5.

`GETEVENT a%()` returns the modifier for a keypress in `(a%(2) AND $ff)`.

❺ `GETEVENT32 ev&()` returns modifiers to `ev&(4)` for keypresses and in `ev&(5)` for pointer (pen) events.

The values which can be returned are as follows:

modifier	*value*	*constant name (Series 5 only)*
Shift	2	`KKmodShift%`
Control	4	`KKmodControl%`
Caps	16	`KKmodCaps%`

❺ For the Series 5 there is additionally a Fn key:

Fn	32	`KKmodFn%`

❸ For the Series 3c there is additionally a Psion key:

Psion	8	-

APPENDIX E

LISTING OF CONST.OPH FOR SERIES 5

```
REM CONST.OPH version 1.10
REM Constants for OPL
REM Last edited on 19 May 1997
REM (C) Copyright Psion PLC 1997

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
CONST KGetMenu%=4150
CONST KKeyUpArrow%=4105
CONST KKeyDownArrow%=4106
CONST KKeyLeftArrow%=4103
CONST KKeyRightArrow%=4104
CONST KKeyPageUp%=4100
CONST KKeyPageDown%=4101
CONST KKeyPageLeft%=4098
CONST KKeyPageRight%=4099

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%=-13
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

CONST KKeyMenu%=4150
CONST KKeySidebarMenu%=10000

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 F

SQL SPECIFICATION FOR THE SERIES 5

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 view, 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 view, in an undefined order; otherwise a comma separated list specifies which columns to return, and the order that the columns appear in the view.

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 view. 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 parentheses 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`.

APPENDIX G

EPOC32 ERROR VALUES

The error values returned by EPOC32 are listed below. Note that these are different from the error values returned by OPL in general. They may, however, be returned in the status word of asynchronous OPX functions.

meaning	*value*	*meaning*	*value*
no error	0	not found	-1
general error	-2	cancel	-3
no memory	-4	not supported	-5
argument error	-6	total loss of precision	-7
bad handle	-8	overflow error	-9
underflow error	-10	already exists	-11
path not found	-12	died	-13
in use	-14	server terminated	-15
server busy	-16	completion	-17
not ready	-18	unknown error	-19
corrupt	-20	access denied	-21
locked	-22	write error	-23
dismounted	-24	end of file	-25
disk full	-26	bad driver	-27
bad name	-28	comms line fail	-29
comms frame error	-30	comms overrun	-31
comms parity error	-32	timed out	-33
could not connect	-34	could not disconnect	-35
disconnected	-36	bad library entry point	-37
bad descriptor	-38	abort	-39
too big	-40	divide by zero	-41
bad power	-42	directory full	-43

INDEX

R

S

T

U