dpNotes are a collection of - hopefully - useful ideas for the furtherance of OPL programming. In dpNotes we focus primarily on Symbian OS v6.0 and later, though some of it was around already at v5 (ER5) and even OPL16 times.

dpNotes

Index

dpNote 0001 - Finding out the number of images in an MBM file
dpNote 0002 - Using the Nokia 9200 Series Communicator system fonts
dpNote 0003 - Getting the path to the location of an OPL application
dpNote 0004 - IOSEEK - documented and undocumented features
dpNote 0005 - Calculating UID checksum
dpNote 0006 - Handling of simple stacks
dpNote 0007 - Multi-dimensional arrays
dpNote 0008 -
dpNote 0009 - Getting the serial (IMEI) number of a Nokia 9200 Series Communicator
dpNote 0010 - Making the v6/S80 WINS Emulator more programmer friendly for OPL development
dpNote 0011 - Showing the amount of free memory
dpNote 0012 - Naming conventions
dpNote 0013 - Conversion between large hexadecimal and decimal numbers
dpNote 0014 - Useful POKEs and PEEKs
dpNote 0015 -
dpNote 0016 - Toolbar buttons with text only
dpNote 0017 - dpToolbar - a better Toolbar for Psion Teklogix netBook and Psion Series 7
dpNote 0018 - Loading a complete file into a buffer
dpNote 0019 - Converting long text buffers between Unicode and Ascii
dpNote 0020 - LOC function which gives correct answer for control and Unicode characters
dpNote 0021 -
dpNote 0022 -
dpNote 0023 -
dpNote 0024 -
dpNote 0025 -
dpNote 0026 -
dpNote 0027 - Launching an application from OPL and wait until it finishes before returning
dpNote 0028 - Asynchronous event loop with inactivity timer
dpNote 0029 - Key event codes for Series 60 phones
dpNote 0030 -

Note 1: We apologise for the number of empty dpNotes still in the table. They do in fact exist in raw form, and will be added as soon as we have the time to do some decent editorial.

Note 2: For ease of reading, we are using `//` comments in lieu of `REM` comments. We only use the latter to inactivate code, not to add comments code. If you try to compile with `//` comments you will of course get an error, but you can easily change to `REM` comments using Find/Replace All.

dpNote 0001

Finding out the number of images in an MBM file

2 July 2002

This procedure is quite useful when the number of images in an MBM file is unknown.

OPL for v5 EIKON

```
CONST KImageCounterOffset&=&00000010
CONST KLongSize&=4
```

OPL for v6.0 Series 80

```
PROC GetNoOfImagesInMbmFile&:(aMbmFileName$)
LOCAL IOstatus%,hMbm%,IOmode%,Offset&,NoOfImages&
// open the image file
IOmode%=KIOModeOpen% OR KIOFormatBinary% OR KIOAccessRandom% OR KIOAccessShare%
IOstatus%=IOOPEN(hMbm%,aMbmFileName$,IOmode%)
IF IOstatus%<0
  RAISE KErrNotExists%
ENDIF
// move to the position of the offset address of the image counter
Offset&=KImageCounterOffset&
IOSEEK(hMbm%,1,Offset&)
// read the position of the image counter
IOREAD(hMbm%,ADDR(Offset&),KLongSize)
// move to the position of the image counter
IOSEEK(hMbm%,1,Offset&)
// read the image counter
```

```
IOREAD(hMbm%,ADDR(NoOfImages&),KLongSize&)
// close the image file
IOCLOSE(hMbm%)
// return number of images in file
RETURN NoOfImages&
ENDP
```

dpNote 0002 **Using the Nokia 9200 Series Communicator system fonts**

5 July 2002

OPL for v6.0 Series 80

The `Const.oph` file for Symbian OS v6.0 does not include the font UIDs for the Nokia 9200 Series Communicator. You could add the following constants to your `Const.oph` file to address this:

```
CONST KFontLindaBold16&=          268457209 // &100054F9
CONST KFontLindaBold18&=          268457210
CONST KFontLindaBold20&=          268457211 // in CBA titles for 9210
CONST KFontLindaBold22&=          268457212
CONST KFontLindaBold24&=          268457213
CONST KFontLindaBold29&=          268457214
CONST KFontLindaBoldItalic20&=    268457215
CONST KFontLindaItalic20&=        268457216
CONST KFontLindaNarrow20&=        268457217
CONST KFontLindaNarrow24&=        268457218
CONST KFontLindaNarrow29&=        268457219
CONST KFontLindaNormal18&=        268457220
CONST KFontLindaNormal20&=        268457221 // in gIPRINT for 9210
CONST KFontLindaNormal24&=        268457222
CONST KFontLindaNormal29&=        268457223 // &10005507

CONST KFontTerminalNormal8&=      268437778 // &10000912, same in ER5
CONST KFontTerminalZoomed15&=     268437779 // &10000913, same in ER5
```

The fonts are now selectable with `gFONT` as normal.

The designations given are now taken from the 9210 screen layout document. They are slightly modified to comply with OPL coding standards.

So where does Linda come from? Who is she? Well, it appears to have been the codename for the Series 80 and/or the Nokia 9200 series.

The `KFontLindaBold20&` UID is particularly useful if you wish to have a consistent font on your CBA titles.

And the `KFontLindaNormal20&` UID is useful if you wish to create variants of `gIPRINT` and `ALERT` messages. By the way, the green frame has the RGB value `KRgbMessageBorderGreen&=&008800`.

dpNote 0003 **Getting the path to the location of an OPL application**

15 July 2002

OPL for v5 EIKON

OPL for v6.0 Series 80

This is a routine used in every OPL application we've developed. We have seen so many different ways of doing the same thing, but we find this method the most universal.

Usage, and one of the first things you do in the application:

```
Location$=GetPathToThisApplication$:
```

```
CONST KMaxFilenameLen%=255
CONST KCmdAppName%=1
CONST KParseFilenameOffset%=4
```

The constants above should normally be included in Const.oph.

```
PROC GetPathToThisApplication$:
LOCAL Path$(KMaxFilenameLen%),Offset%(6)
Path$=PARSE$(CMD$(KCmdAppName%),"",Offset%())
Path$=LEFT$(Path$,Offset%(KParseFilenameOffset%)-1)
RETURN Path$
ENDP
```

dpNote 0004 **IOSEEK - documented and undocumented features**

15 July 2002

OPL for v5 EIKON

OPL for v6.0 Series 80

`IOSEEK` has two more uses than the OPL manuals, for instance the OPL Guide & Reference (Release 036 for ER5), usually mention. Modes 4 and 5 will return or set the current position in a text file. Furthermore, Mode 3 will return the resulting pointer in a binary file.

The typical usage is `IOSEEK(hFile%,IOmode%,Offset&)`, where `hFile%` is the relevant handle returned by `IOOPEN`, and `IOmode%` is the `IOSEEK` Mode. The `Offset&` is used to specify or receive pointers and positions.

`IOmode%=1` will set the pointer in a binary file to the absolute value specified in `Offset&`.

`IOmode%=2`, will set the pointer in a binary file to `Offset&` bytes from the end of the file, i.e. `Offset&` is here equal to `FileSize&-AbsoluteOffset&`.

`IOmode%=3` will set the pointer in a binary file to `Offset&` bytes relative to the current position. At the same time, `Offset&` will also be set to the resulting pointer. In other words, if `Offset&=0`, `IOSEEK(hFile%,3,Offset&)` will make `Offset&` take the value of the current pointer. This is sometimes not clearly documented either.

`IOmode%=4` will set `Offset&` to the current position in a text file.

`IOmode%=5` will set the current position in a text file to `Offset&`.

`IOmode%=6` will rewind a text file to the first position. `Offset&` is not used, but you must still pass it as a argument.

Take note that by convention we talk about 'pointers' when handling binary files (Modes 1-3) and 'positions' when handling text files (Modes 4-6).

dpNote 0005 — Calculating UID checksum

15 July 2002

OPL for v6.0 Series 80

Want to define your own file formats? You will need to extract the UID4 from the `SyUIDCheckSum$:` function. But this function, though you are passing three long integers to it, returns a string. And this string contains the other UIDs as well. This procedure helps you extract the UID4 as a long integer.

```
INCLUDE "System.oxh"

PROC UidCheckSumCalculate&:(aUID1&,aUID2&,aUID3&)
LOCAL Header$(16),UIDNo%,UID&
// the System.opx function returns the complete file header as a string
Header$=SyUIDCheckSum$:(aUID1&,aUID2&,aUID3&)
// since we entered LongInt values, we should return LongInt values
// (N.B. this does not work in ER5 due to Ascii)
// by setting this to 1,2,3 the other UIDs can be returned
// but this is of course not needed here
UIDNo%=4
UID&=PEEKL(ADDR(Header$)+KUnicodeHeader%+(UIDNo%-1)*4)
RETURN UID&
ENDP
```

dpNote 0006 — Handling of simple stacks

15 July 2002

OPL for v6.0 Series 80

This dpNote contains some general routines for stack handling. They can be used as a framework to build more advanced data formats, e.g. multidimensional arrays, lists and trees etc, as OPL does not have inherent support for these. Here, the stack routines are written to handle short integer values. They can easily be modified to handle values of other byte sizes.

```
CONST KStackHeader%=4
CONST KShortSize&=2

PROC StackCreate&:
// Usage: pStack&=StackCreate&
LOCAL p&
p&=ALLOC(KStackHeader%)
// 0 items so far
POKEL p&,&0000
RETURN p&
ENDP

PROC StackDestroy:(apStack&)
FREEALLOC apStack&
ENDP

PROC StackEmptyB%:(apStack&)
// returns KTrue% if stack is empty
RETURN (PEEKL(apStack&)=&0000)
ENDP

PROC StackDepth&:(apStack&)
// returns the number of items in the stack
RETURN PEEKL(apStack&)
ENDP

PROC StackAddr&:(apStack&,aNo&)
// returns the pointer address of a stack item number aNo&
IF aNo&<1 OR aNo&>StackDepth&:(apStack&)
  RAISE KErrInvalidStackNo%
ENDIF
```

```
RETURN apStack&+KShortSize&*(aNo&-1)+KStackHeader%
ENDP

PROC StackRecall%:(apStack&,aNo&)
// returns the value of the item number aNo&
// usage: Value%=StackRecall%:(pStack&,Number%)
RETURN PEEKW(StackAddr&:(apStack&,aNo&))
ENDP

PROC StackStore:(apStack&,aNo&,aValue%)
// stores the value aValue% at position aNo&
POKEW StackAddr&:(apStack&,aNo&),aValue%
ENDP

PROC StackPeek%:(apStack&)
// returns the value of the item on top of the stack
// usage: Value%=StackPeek%:(pStack&)
RETURN PEEKW(StackAddr&:(apStack&,StackDepth&:(apStack&)))
ENDP

PROC StackPush&:(apStack&,aValue%)
// usage: pStack&=StackPush&:(pStack&,Value%)
LOCAL pNew&,size&,NewNoItems&
// calculate the new no of items
NewNoItems&=1+StackDepth&:(apStack&)
// calculate the new cell size
size&=KStackHeader%+KShortIntWidth&*NewNoItems&
// adjust to new cell size
pNew&=REALLOC(apStack&,size&)
IF pNew&=0
  RAISE KErrNoMemory%
ENDIF
// insert resulting number of items
POKEL pNew&,NewNoItems&
// insert value
POKEW StackAddr&:(pNew&,NewNoItems&),aValue%
// return new pointer
RETURN pNew&
ENDP

PROC StackPop%:(apStack&)
// usage: Value%=StackPop%:(pStack&)
LOCAL Value%,OldNoItems&,pNew&,size&,NewNoItems&
IF StackEmptyB%:(apStack&)
  RAISE KErrStackEmpty%
ENDIF
OldNoItems&=StackDepth&:(apStack&)
Value%=PEEKW(StackAddr&:(apStack&,OldNoItems&))
// calculate the new no of items
NewNoItems&=OldNoItems&-1
// calculate the new cell size
size&=KStackHeader%+KShortIntWidth&*NewNoItems&
// adjust to new cell size
pNew&=REALLOC(apStack&,size&)
// this check should not be needed, but let's check it anyway
IF pNew&<>apStack&
  RAISE KErrShrinkCellFailure%
ENDIF
// insert resulting number of units
POKEL pNew&,NewNoItems&
// return popped value
RETURN Value%
ENDP

PROC StackClear:(apStack&)
// guarantees that the pointer value is the same,
// which StackDestroy: followed by StackCreate&: would not
LOCAL pNew&
// shrinking the cell
pNew&=REALLOC(apStack&,KStackHeader%)
// this check should not be needed, but let's check it anyway
IF pNew&<>apStack&
  RAISE KErrShrinkCellFailure%
ENDIF
// zero the counter
```

```
POKEL apStack&,&0000
ENDP

PROC StackDelete:(apStack&,aNo&)
// deletes an item without changing the order of other items
LOCAL OldNoOfItems&,Item&
OldNoOfItems&=StackDepth&:(apStack&)
// copy all items above the deleted item one step downwards
Item&=aNo&
WHILE Item&<OldNoOfItems&
  StackStore:(apStack&,Item&,StackRecall%:(apStack&,Item&+1))
  Item&=Item&+1
ENDWH
// remove the last item
StackPop%:(apStack&)
ENDP

PROC StackSwap:(apStack&,aNo1&,aNo2&)
LOCAL tmp%
tmp%=StackRecall%:(apStack&,aNo1&)
StackStore:(apStack&,aNo1&,StackRecall%:(apStack&,aNo2&))
StackStore:(apStack&,aNo2&,tmp%)
ENDP

PROC StackMirror:(apStack&)
// turns the stack inside-out so that what was previously on top,
// is now at the bottom etc
LOCAL LastNo&,MiddleNo&,q&,tmp%
IF StackEmptyB%:(apStack&)
  RAISE KErrStackEmpty%
ENDIF
LastNo&=StackDepth&:(apStack&)
// no point doing stack mirror if no of items less than 2
IF LastNo&<2
  RETURN
ENDIF
// doesn't actually matter if LastNo& is even or odd
MiddleNo&=LastNo&/2
q&=1
DO
  StackSwap:(apStack&,q&,LastNo&+1-q&)
  q&=q&+1
UNTIL q&>MiddleNo&
ENDP

PROC StackOccurrences%:(apStack&,aValue%)
// returns number of occurrences of a value
LOCAL found%,i&,NoOfItems&
found%=0
NoOfItems&=StackDepth&:(apStack&)
IF NoOfItems&=0
  RETURN found%
ENDIF
i&=1
DO
  IF aValue%=StackRecall%:(apStack&,i&)
    found%=found%+1
  ENDIF
  i&=i&+1
UNTIL i&>NoOfItems&
RETURN found%
ENDP

PROC StackFind&:(apStack&,aValue%)
// returns position of first occurrence of a value counted from top
// returns 0 if not found
LOCAL position&,NoOfItems&
position&=0
IF StackEmptyB%:(apStack&)
  RETURN position&
ENDIF
NoOfItems&=StackDepth&:(apStack&)
position&=NoOfItems&
DO
  IF aValue%=StackRecall%:(apStack&,position&)
```

```
    BREAK
  ENDIF
  position&=position&-1
UNTIL position&=0
RETURN position&
ENDP
```

dpNote 0007

Multi-dimensional arrays

28 Aug 2002
(Updated
4 Dec 2003)

OPL for v6.0
Series 80

OPL keywords do not support multidimensional arrays. But it is reasonably easy to implement it using the stack model of dpNote 0006. In this example we are looking at some procedures for handling integer two dimensional arrays.

The code is calling procedures and using constants of the dpNote 0006.

```
PROC ArrayDim&:(aNoOfColumns%,aNoOfRows%)
// this procedure sets up the array and remembers its dimension - similar to the
// DIM command of some BASIC dialects. Just remember that Column comes before Row,
// just as x comes before y.
// Usage pMyArray&=ArrayDim&:(NumberOfColumns%,NumberOfRows%)
LOCAL pArray&,TotalNoOfPositions%,i%
// calculate the total number of positions
TotalNoOfPositions%=aNoOfColumns%*aNoOfRows%
// create stack
pArray&=StackCreate&:
pArray&=StackPush&:(pArray&,aNoOfColumns%) // remember no of columns in 1st header
pArray&=StackPush&:(pArray&,aNoOfRows%) // remember no of rows in 2nd header
// initialise the complete array to zero. You could also use REALLOC for this,
// but that would not guarantee that all variables in the array are set to zero
i%=1
WHILE i%<=TotalNoOfPositions%
  pArray&=StackPush&:(pArray&,$0)
  i%=i%+1
ENDWH
// return the pointer
RETURN pArray&
ENDP

PROC ArrayGetCols%:(apArray&)
RETURN StackRecall%:(apArray&,&0001)
ENDP

PROC ArrayGetRows%:(apArray&)
RETURN StackRecall%:(apArray&,&0002)
ENDP

PROC ArrayPos&:(apArray&,aCol%,aRow%)
// returns the position of the element of an array item (aCol%,aRow%)
LOCAL Position&,TotalCol%,TotalRow%
// obtain the array size
TotalCol%=ArrayGetCols%:(apArray&)
TotalRow%=ArrayGetRows%:(apArray&)
// check that the position is valid, and give error if otherwise
IF aCol%<1 OR aCol%>TotalCol% OR aRow%<1 OR aRow%>TotalRow%
  RAISE KErrInvalidStackNo%
ENDIF
// return position and add 2 for the header
Position&=(aRow%-1)*TotalCol%+aCol%+2
RETURN Position&
ENDP

PROC ArrayAddr&:(apArray&,aCol%,aRow%)
// returns the pointer address of the array item (aCol%,aRow%)
RETURN StackAddr&:(apArray&,ArrayPos&:(apArray&,aCol%,aRow%))
ENDP

PROC ArraySto:(apArray&,aCol%,aRow%,aValue%)
// stores the value aValue% to array item (aCol%,aRow%)
StackStore:(apArray&,ArrayPos&:(apArray&,aCol%,aRow%),aValue%)
ENDP

PROC ArrayRcl%:(apArray&,aCol%,aRow%)
// returns the value of the array item (aCol%,aRow%)
RETURN StackRecall%:(apArray&,ArrayPos&:(apArray&,aCol%,aRow%))
ENDP
```

```
PROC ArrayDestroy:(apArray&)
// erases array and frees up memory
StackDestroy:(apArray&)
ENDP
```

Assume now that we wish to create and populate a matrix A of 5 colums and 2 rows and a matrix B of 2 columns and 3 rows, and then do some calculations, this is how it can be done.

```
CONST KMatrixACol%=5
CONST KMatrixARow%=2
CONST KMatrixBCol%=2
CONST KMatrixBRow%=3

PROC Main:
LOCAL A&,B& // matrix handlers
// initialise arrays
A&=ArrayDim&:(KMatrixACol%,KMatrixARow%)
B&=ArrayDim&:(KMatrixBCol%,KMatrixBRow%)
// populate array A with '25' in all positions
ArrayPopulate:(A&,25)
// populate array B with '13' in all positions
ArrayPopulate:(B&,13)
// add A(5,2) to B(2,3) and put in A(1,1) and print it, it should print 38
ArraySto:(A&,1,1,ArrayRcl%:(A&,5,2)+ArrayRcl%:(B&,2,3))
PRINT ArrayRcl%:(A&,1,1)
GET
// erase both arrays and release memory
ArrayDestroy:(A&)
ArrayDestroy:(B&)
ENDP

PROC ArrayPopulate:(apArray&,aValue%)
LOCAL i%,j%,TotalCol%,TotalRow%
TotalCol%=ArrayGetCols%:(apArray&)
TotalRow%=ArrayGetRows%:(apArray&)
i%=1
WHILE i%<=TotalCol%
  j%=1
  WHILE j%<=TotalRow%
    ArraySto:(apArray&,i%,j%,aValue%)
    j%=j%+1
  ENDWH
  i%=i%+1
ENDWH
ENDP
```

It would now also be easy to write procedures for matrix addition and multiplication etc, where the result is put in matrix C.

```
C&=ArrayAdd&:(A&,B&) // should give error if A& and B& are not of same size
C&=ArrayAddValue&:(A&,aValue%)
C&=ArrayMultiply&:(A&,B&) // assumes no of colums in A& = no of rows in B&
C&=ArrayMultiplyByValue&:(A&,aValue%)
```

But we won't do it this time.

dpNote 0009 **Getting the serial (IMEI) number of a Nokia 9200 Series Communicator**

28 Aug 2002

OPL for v6.0
Series 80

The serial number is the one normally written below the barcode and it only becomes visible once you remove the battery. Of course you can also get it from the phone screen by pressing *#06#. The number is also referred to as the IMEI number.

```
INCLUDE "System.oxh"

CONST KWINSSerialNumber$="111111111111111"

PROC SerialNumberGet$:
// first check if the routine is run in WINS, because there is
// a bug in SyGetPhoneInfo$:, it hangs when run from emulator
IF MachineTypeGet&:<>KSyMachineUid_Win32Emulator&
  // SyGetPhoneInfo$: returns a 16 character string. The rightmost
  // character is unknown. It needs to be truncated off
  // to avoid any issues with later comparisons.
  RETURN LEFT$(SyGetPhoneInfo$:(KPhoneInfoSerialNumber%),15)
```

```
ELSE
  // if run from WINS, return a faked serial number
  RETURN KWINSSerialNumber$
ENDIF
ENDP

PROC MachineTypeGet&:
LOCAL value&,return&
value&=0
return&=SyGetHAL&:(KSyMachineUID&,value&)
RETURN value&
ENDP
```

dpNote 0010

Making the v6/S80 WINS Emulator more programmer friendly for OPL development

3 Sep 2002

OPL for v6.0 WINS

We find it most practical to program OPL in the WINS Emulator. In WINS for v5.0 there was a "Psion 5mx/netBook-style" VGA console. As far as we are concerned using this console provides the best trade-off between overview and ability to view other windows on the PC screen at the same time. Moreover, the width is 640 pixels, exactly like the Nokia Communicator's screen. But the height is 480 pixels in lieu of the Communicator's 200 pixels.

Once you have installed the WINS emulator on a PC, usually the WINS console layout is defined in the `C:\Symbian\6.0\NokiaCPP\Epoc32\Data` folder. This is done in the `epoc.ini` file. The console image used is defined by the statements:

```
# screen size and location
ScreenWidth 640
ScreenHeight 200
ScreenOffsetX 137
ScreenOffsetY 62
fasciabitmap 9210Small.bmp
```

Now, you can change these values to fit the WINS v5 VGA console.

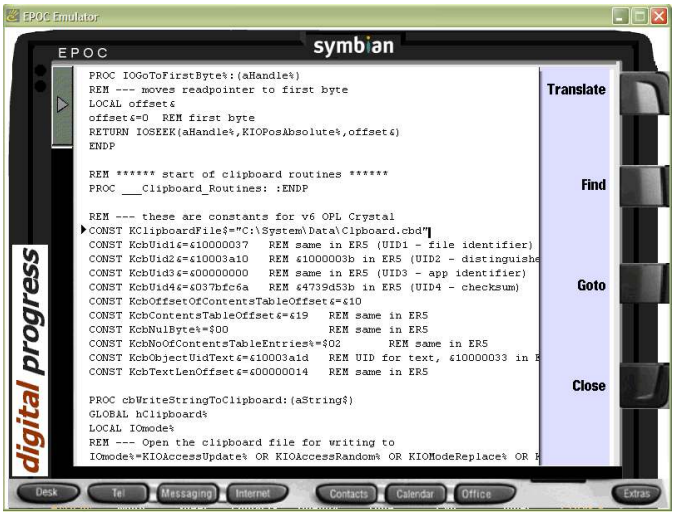
```
# screen size and location
ScreenWidth 640
ScreenHeight 480
ScreenOffsetX 87
ScreenOffsetY 48
fasciabitmap VGA9210.bmp
```

The first time we did it, we noted that some things in the Series 80 (a.k.a. Crystal) UI are hardcoded and some are softcoded. For instance the status bar to the left seems hardcoded, as the battery symbol etc does not move to the bottom. But the CBA to the right seems softcoded, so the buttons spread out over the available height. This requires some tweaking of the `epoc.ini` file so as to fit the CBA button areas to their new locations. And the CBA buttons will need to be located correctly in the image, since they are literally "hardcoded" on a Communicator and not like the Toolbar of v5.0 vintage.

Furthermore, the "Psion 5mx-style" silkscreen buttons below the screen have to be replaced by the Series 80 application buttons. This requires pasting of the application button image and aligning the corresponding parameters in the `epoc.ini` file to the new location.

And lastly, almost all the Symbian OS v5.0 machines, including the netPad, have silkscreen buttons to the left, which the Communicator does not have. Well, we masked them out, except for the top one [Menu], which we found practical to keep; it makes it easier to access the Menu than to hit F1. This also means that definition of a new virtual key is needed.

It all becomes a "hybrid" looking creation, but we find it very practical.



If anyone wish to do this, the console image file and the epoc.ini file are downloadable below. They should be put in `C:\Symbian\6.0\NokiaCPP\Epoc32\Data`.

Remember to backup or rename the existing epoc.ini file before you replace it. At the end of your OPL development you will probably want to go back and test your application with a Nokia 9200 Series Communicator console.

Console image file: VGA9210.bmp
Console settings file: epoc.ini.

dpNote 0011 **Showing the amount of free memory**

3 Sep 2002

OPL for v6.0 Series 80

```
INCLUDE "System.oxh"
INCLUDE "Const.oph"

PROC MessageFreeMemory:
Message:("Free RAM: "+GEN$(FreeMemoryGet&:/1024/1024,12)+" MBytes")
ENDP

PROC Message:(aMsg$)
gIPRINT aMsg$,KBusyTopRight%
ENDP

PROC FreeMemoryGet&:
LOCAL TotalRam&,TotalRom&,MaxFreeRam&,FreeRam&
SyMemoryInfo:(TotalRam&,TotalRom&,MaxFreeRam&,FreeRam&)
RETURN FreeRam&
ENDP
```

dpNote 0012 **Naming conventions**

4 Sep 2002

OPL for v5 EIKON

OPL for v6.0 Series 80

OPL for v6.1 Series 60

Conventions are used to make coding easier. The larger the program, the more sense they make. These are some of our naming conventions for constants, variable names, procedure names etc that we use. Adopt it or leave it. We have seen many other conventions from other developers which also make sense. Whichever convention you use, it is helpful.

Constants should commence with a capital 'K', e.g. `KMaxStringLen%`. Some OPL documentations say that constants can only be used in OPL file headers and in OPH files. In fact, they can be used anywhere in the code outside a procedure as long as they are declared before they are used. Therefore, constants can be "local" to a procedure. Just make sure that their names do not clash with other constants.

Globals should commence with a capital 'G', e.g. `GScreenWidth%=gWIDTH`. Using this convention helps you remember which variables should be declared with the `EXTERNAL` statment in procedures. It also helps you reducing the use of globals.

Variables passed by value should commence with an 'a' in the receiving procedure, e.g.

```
PROC LoadTextFileIntoBuffer&:(aFileName$)
LOCAL pBuffer&
// do it
RETURN pBuffer&
ENDP

could be called by pMyTextBuffer&=LoadTextFileIntoBuffer&:("MyTextFile.txt")
```

Variables passed by reference should commence with an '_' in the receiving procedure. In this procedure the complex number Y is passed by value and added into the complex number X which is passed by reference and later returned by reference.

```
PROC ComplexAdd:(_ReX&,_ImX&,aReY,aImY)
LOCAL ReX,ImX
// acquire values passed by reference
ReX=PEEKF(_ReX&) :ImX=PEEKF(_ImX&)
// make complex number addition
ReX=ReX+aReY :ImX=ImX+aImY
// return values
POKEF _ReX&,ReX :POKEF _ImX&,ImX
ENDP
```

The procedure could be called by: `ComplexAdd:(ADDR(ReZ),ADDR(ImZ),-2.7,4.3)` , after which ReZ and ImZ will have the resulting value.

Boolean variables, i.e. variables that should only take the value `KTrue% or KFalse% should` end with a 'B', e.g. `GCbaVisibleB%`. In this case it is also a global variable. This is also recommended if a procedure can only return `KTrue% or KFalse%` e.g. `TitleBarVisibleB%:`.

Pointers should commence with a 'p', e.g. `pStack&=ALLOC(KStackHeader%)`

Handles, for instance when using IO operations, should commence with a 'h' as in `IOOPEN(hInfile%,aFileName$,IOmode%)`.

Window IDs, should end with 'WID', e.g. `TitleWID%=gCREATE(0,0,TitleWidth%,ScreenHeight%,KVisible%,K4KColourMode%)`

Variables that are used for application and file **UIDs** should end with 'UID', e.g. `ApplicationUID&`

Other types of IDs, e.g. for handling of sprites, should end with ID, e.g. `SpriteID&=SpriteCreate&:(TitleWID%,KHiddenPosXY%,KHiddenPosXY%,KFalse%)`

Labels that are called from `GOTO` and `ONERR` and the like should commence with 'LBL_', e.g. `ONERR LBL_Cleanup::`.

Not only as a nod to object oriented programming, but also because makes it easier to navigate in large programs, it is better to name procedures and variables in "inverted language", meaning that you commence with the "class name", and then, if applicable, indicate the "sub class" and end with specifying your action. For instance, it makes it easier to name your clipboard handling procedure as `ClipboardStringWrite:(aString$)` rather than `WriteStringToClipboard:(aString$)`. The former is easier to find, though of course the latter flows better as human language.

dpNote 0013

4 Sep 2002 (Modified 25 October 2004)

OPL for v5 EIKON

OPL for v6.0 and v7.0s Series 80

OPL for v6.1 and v7.0s Series 60

OPL for v7.0 UIQ

Conversions between large Hexadecimal and Decimal numbers

The typical way to obtain a string with hexadecimal representation of a number is `HEX$(&nnnnnnnn)`. This only works for hexadecimal numbers up to `&ffffffff`.

Converting to decimal representation of a hexadecimal number can be done via `EVAL("&"+"nnnnnnnn")`, but this method will only work up to `&3fffffff` since the `EVAL` function interpretes a the `&nnnnnnnn` expression as a signed hexadecimal number.

The following two procedures will work with up to 16 digits in decimal format and up to 13 digits in hexadecimal unsigned format. An additional advantage is that they both receive and return the numbers as strings.

Take note that, as the procedures are written here, they will not accept any impurities, e.g. strings with signs, points or exponents, so these elements will have to be cleared out beforehand.

```
PROC CvHexFromDec$:(aDecString$)
// safe to use for up to 13 significant hex digits
// and 16 significant dec digits
LOCAL HexString$(KMaxStringLen%)
LOCAL value,residue
// convert from lowest significant while shifting downwards
value=EVAL(aDecString$)
HexString$=""
DO
  value=value/16.0
  residue=(value-INTF(value))*16.0
  value=INTF(value)
  HexString$=HEX$(residue)+HexString$
UNTIL value=0
IF LEN(HexString$)>13
  RAISE KErrOverflow%
ENDIF
RETURN HexString$
ENDP

PROC CvDecFromHex$:(aHexString$)
// safe to use for up to 13 significant hex digits
// and 16 significant dec digits
LOCAL value,i%,len%
len%=LEN(aHexString$)
IF len%<=7
  // if HexString smaller than 8 digits a faster method can be used
  value=EVAL("&"+aHexString$)
ELSEIF len%<=13
  // convert one by one and shift upwards
  i%=len%
  value=0
  DO
    value=value*16.0+EVAL("$"+MID$(aHexString$,len%-i%+1,1))
    i%=i%-1
```

```
  UNTIL i%<1
ELSE
  RAISE KErrOverflow%
ENDIF
RETURN GEN$(value,16)
ENDP
```

dpNote 0014 **Useful POKEs and PEEKs**

12 Aug 2005

dpNote0014 has some useful POKE and PEEK operations

OPL for v5 EIKON

OPL for v6.0 and v7.0s Series 80

OPL for v6.1 and v7.0s Series 60

OPL for v7.0 UIQ

```
PROC PokeBoolean:(aPointer&,aBooleanB%)
// used to reduce space needed for a boolean value in a descriptor
// KTrue% will be stored as $ff and KFalse% will be stored as $00
POKEB aPointer&,(aBooleanB% AND $ff)
ENDP

PROC PeekBooleanB%:(aPointer&)
// used to PEEK a boolean value stored with PokeBoolean
RETURN NOT (PEEKB(aPointer%)=$00)
ENDP

PROC PokeCharBE:(aPointer&,aChar16&)
// POKEs a 2 byte Unicode character as big endian
POKEB aPointer&,((aChar16& AND &ff00)/&0100)
POKEB (aPointer&+&1),(aChar16& AND &00ff)
ENDP

PROC PeekCharBE&:(aPointer&)
// PEEKs a 2 byte Unicode character as big endian
RETURN (PEEKB(aPointer&)*&0100) OR PEEKB(aPointer&+&1)
ENDP

PROC PokeCharLE:(aPointer&,aChar16&)
// POKEs a 2 byte Unicode character as little endian
POKEB aPointer&,(aChar16& AND &00ff)
POKEB (aPointer&+&1),((aChar16& AND &ff00)/&0100)
ENDP

PROC PeekCharLE&:(aPointer&)
// PEEKs a 2 byte Unicode character as little endian
RETURN PEEKB(aPointer&)OR (PEEKB(aPointer&+&1)*&0100)
ENDP
```

dpNote 0016 **Toolbar buttons with text only**

6 Sep 2004

OPL for v5.0 EIKON

The following code is a workaround for creating Toolbar buttons without a bitmap. If a bitmap is not declared when using Toolbar this causes empty boxes to be displayed on devices like Psion Series 5 and Psion netBook. The trick is to use an empty bitmap.

```
PROC BitMapZeroCreate&:
// Returns handle to empty bitmap to be used in TBarButt
LOCAL ZeroBM&
// clearer to go via variable, coz gCREATEBIT returns short int
ZeroBM&=gCREATEBIT(0,0)
RETURN ZeroBM&
ENDP
```

and then:

```
PROC main:
GLOBAL GZeroBM&
.
.
GZeroBM&=BitMapZeroCreate&:
.
.
// in toolbar initialisation routine
TBarButt:("a",1,"Welcome",0,GZeroBM&,GZeroBM&,0)
.
.
// in exit routine
gCLOSE GZeroBM&
```

dpNote 0017

15 March 2004

(updated 11 August 2004)

OPL for v5.0 EIKON (only Psion Teklogix netBook and Psion Series 7)

dpToolbar - a better Toolbar for Psion Teklogix netBook and Psion Series 7

[Nota bene] *We might have our good old Toolbar back again in forthcoming Series 90 phones, including OPL support. The default Series 90 style guide supports 3 Toolbar buttons a la Revo without bitmaps.*

dpToolbar is, what we hope, an improved Toolbar for the VGA colour screen Psions. It is meant to be used as an OPL programming tool. It adds the following features compared to the built in Toolbar:

1. Fix of the battery symbol bug - the battery symbol is now drawn, so no MBMs are needed.

2. Fix of the 6th button latch bug. In the orginal Toolbar, the 6th button does not latch properly. Now it does.

3. Ability to toggle the toolbar between left and right side of screen. In fact this feature was the original reason why we developed dpToolbar.

4. Fix of the undefined button selection bug - which previously could create hanging

5. Battery background turns red when its time to change backup battery. With the original Toolbar, nothing actually happens with the battery symbol when the backup battery needs to be replaced, even though one can see from the code that some action was intended. Now the background around the battery symbol turns red when the backup battery needs to be replaced.

6. When clicking on the battery symbol on the dpToolbar, the system battery window will instantly be brought to the foreground.

7. The battery is updated with higher precision than the original Toolbar, the latter only having four levels.

8. dpToolbar is now fully backwards compatible with the original Toolbar.

9. dpToolbar is now fully compatible with Andrew Gregory's Toolbar Patcher.

10. dpToolbar can now display Robin Hood's MoonClock. It will detect if MoonClock is installed, locate the mbm-file and display the phase of the moon based on the current date.

11. A small yellow arrow is now shown when external power is connected.

12. In the original Toolbar, only some actions will trigger a battery status update. Now all actions will.

dpToolbar battery status

 The main battery is full and the external power source is connected.

 The main battery is empty and there is no external power source.

 The main battery has been removed, but the external power source is connected.

 The main battery has been reinserted and is recharging from external power.

 The backup battery is empty or removed. Insert a fresh backup battery as soon as possible.

Location and use of dpToolbar files

dpToolbar comprises of two files.

1. `Toolbar.opo` should be placed into `X:System\Opl\`, where `X` can be either `C` or `D`.

2. `Toolbar.oph` should always be placed in `X:System\Opl\`, where `X` can be either `C` or `D`.

You must also modify the include statement in the application to:

```
INCLUDE "Toolbar.oph"
```

dpToolbar can now be used exactly like the original Toolbar. However to cover all options you should use the following routine to load dpToolbar:

```
CONST KPathAlternative1$="C:\System\Opl\Toolbar.opo"
CONST KPathAlternative2$="D:\System\Opl\Toolbar.opo"
CONST KPathAlternative3$="E:\System\Opl\Toolbar.opo"

IF EXIST(KPathAlternative1$)
  Message:("Loaded: "+KPathAlternative1$)
  LOADM KPathAlternative1$
ELSEIF EXIST(KPathAlternative2$)
  Message:("Loaded: "+KPathAlternative2$)
  LOADM KPathAlternative2$
```

```
ELSEIF EXIST(KPathAlternative3$)
  Message:("Loaded: "+KPathAlternative3$)
  LOADM KPathAlternative3$
ELSE
  Message:("Toolbar.opo not found","Error in loading","See instructions! ")
  PAUSE 40 :STOP
ENDIF
TBarLink:("Main")
```

OPXs

dpToolbar will also need to have `Date.opx`, `System.opx`, `SysRam.opx` and `SystInfo.opx` installed.

Public Procedures

The following is a list of public procedures of dpToolbar. There are some additions.

```
// Public procedures
EXTERNAL TBarLink:(aAppLink$)
EXTERNAL TBarInit:(aTitle$,aScreenW%,aScreenH%)
EXTERNAL TBarInitC:(aTitle$,aScreenW%,aScreenH%,aWindowMode%)
EXTERNAL TBarInitNonStd:(aName$,aScreenW%,aScreenH%,aWidth%)
EXTERNAL TBarSetTitle:(aName$)
EXTERNAL TBarButt:(aShortCut$,aPos%,aText$,aState%,aBitmap&,aMask&,aFlags%)
EXTERNAL TBarOfferB%:(aWID&,aPtrType&,aPtrX&,aPtrY&)
EXTERNAL TBarLatch:(aComponent%)
EXTERNAL TBarShow:
EXTERNAL TBarHide:
EXTERNAL TBarOrientationLeft:                    REM // added in dpToolbar
EXTERNAL TBarOrientationRight:                   REM // added in dpToolbar
EXTERNAL TBarColour:(aFgR%,aFgG%,aFgB%,aBgR%,aBgG%,aBgB%)
EXTERNAL TBarIndicators:                         REM // rewritten in dpToolbar
EXTERNAL TBarDpVersion$:                         REM // added in dpToolbar
```

`TBarOrientationLeft:` will orient the Toolbar to the left side of the screen instead of the right. This is of importance for some work situations when the right hand is occupied or when the overall screen layout so demands it. Several of DP applications, for instance dpCalc, make use of this function.

`TBarOrientationRight:` will orient the Toolbar back to the right side of the screen.

`TBarIndicators:` will now update both the battery information and check the backup battery.

Public Variables

The following globals are available for the application:

`TbWidth%` holds the pixel width of the toolbar.

`TbVisibleB%` carries the value `KTrue%` if visible and otherwise `KFalse%`. Take note that according to DP coding norms, a boolean variable name ends with `B%`. This is a change compared to the original Toolbar.

`TbRightB%` has the value `KTrue%` if the Toolbar is right oriented and `KFalse%` if it is left oriented. This is a new variable. `TBarOrientationLeft:` and `TBarOrientationRight:` will automatically set `TbRightB%` accordingly, so `TbRightB%` should only be **tested**, not modified, by the application.

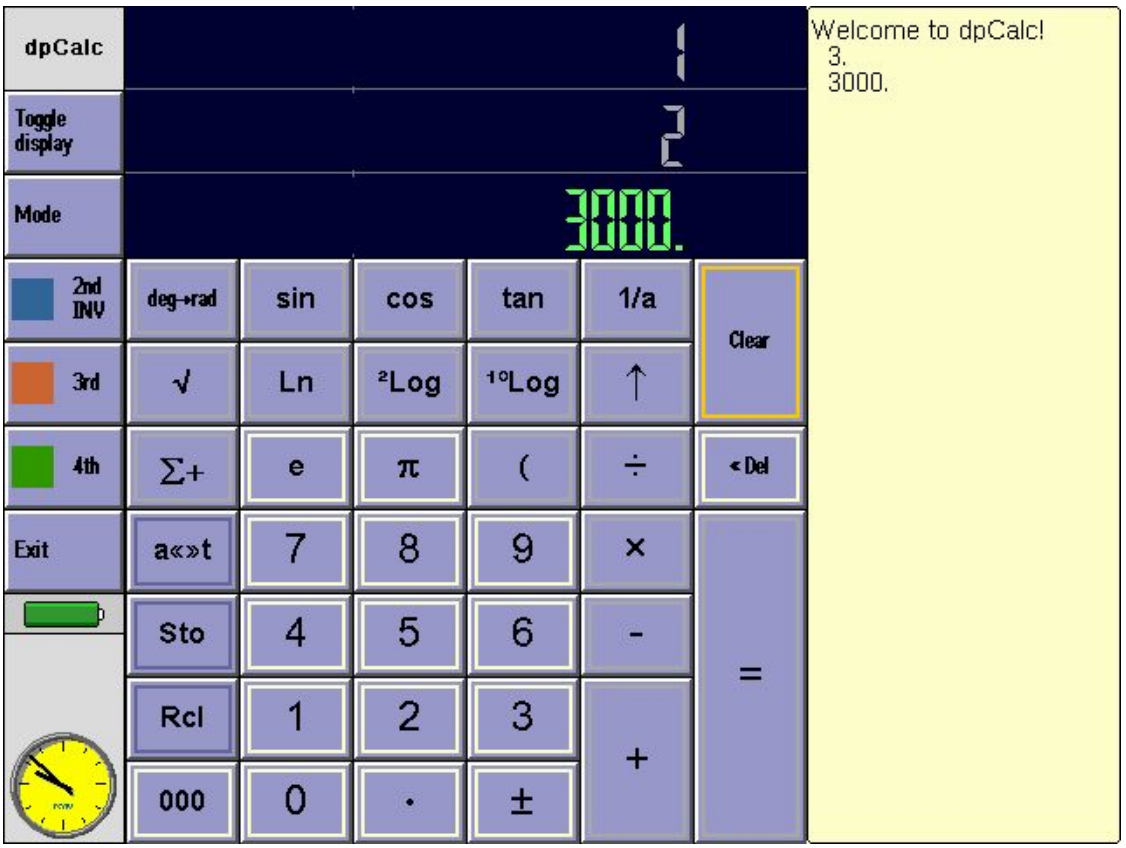
`TbMenuSym%` carries the current 'Show toolbar' menu symbol (to be OR:ed with shortcut letter). This has not been changed since the original Toolbar.

`TbWID%` holds the Toolbar Window ID. Take note of the name change. DP always uses `WID%` to denote a window ID variable.

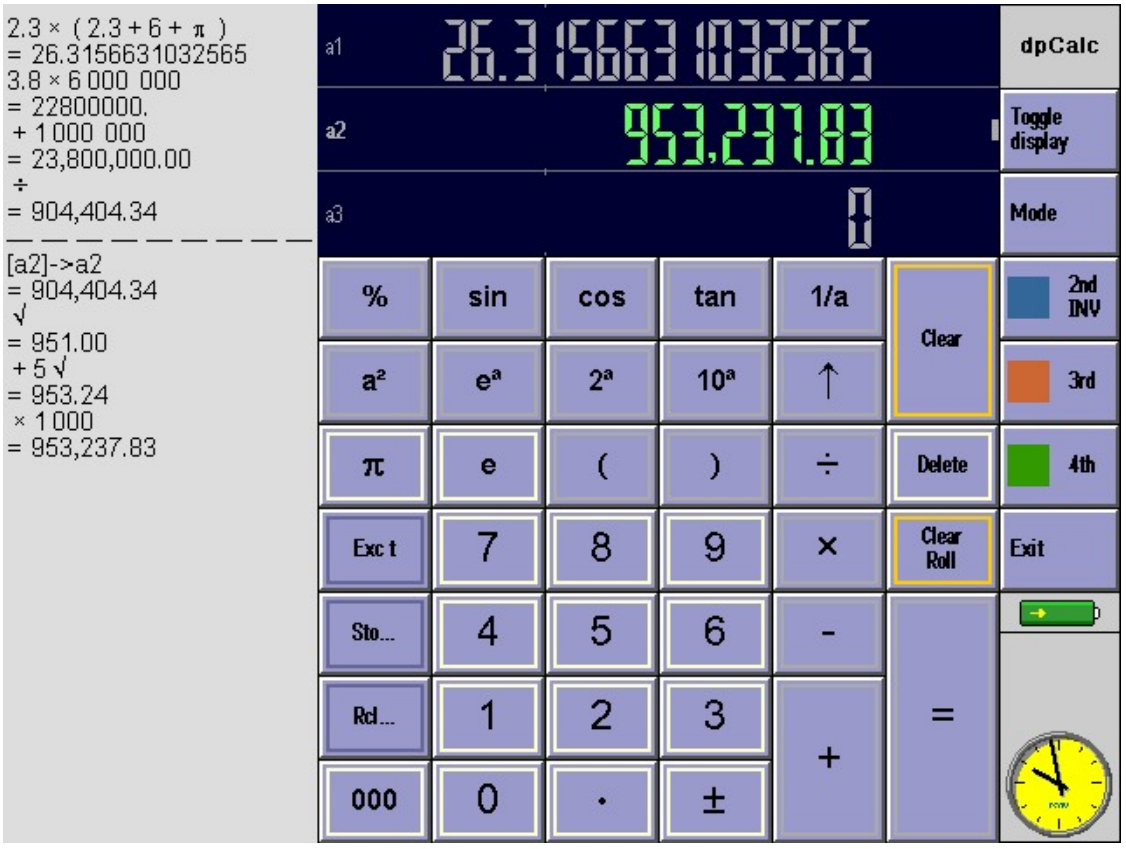
Files and demo application

The files are downloadable here together with a simple demonstration application. Running the demo application, pressing Ctrl+T will toggle between left and right.

This is dpCalc with the Toolbar oriented to the left. The external power is OFF.



This is dpCalc with the Toolbar oriented to the right. The external power is ON:



dpNote 0018 **Loading a complete file into a buffer**

20 Dec 2002

This procedure will load a complete file into a buffer, id est into memory, for further manipulation.

OPL for v6.0
Series 80

Usage:

```
pOutBuffer&=TextFileLoadIntoBuffer&:(Path$+FileName$)

INCLUDE "System.oxh"
INCLUDE "Const.oph"

CONST KReadChunk&=&2000
```

```
PROC TextFileLoadIntoBuffer&:(aFileName$)
LOCAL pBuffer&,InFilesize&
LOCAL hInfile%,IOmode%,r%,IOreturn&
// allocate buffer according to file size
InFilesize&=SyFileSize&:(aFileName$)
Message:("Filesize: "+GEN$(InFilesize&,12))
pBuffer&=ALLOC(InFilesize&+KReadChunk&) // some hanging space here
// load text file into the buffer
IOmode%=KIOModeOpen% OR KIOFormatBinary% OR KIOAccessRandom%
IOOPEN(hInfile%,aFileName$,IOmode%)
r%=0
DO
  IOreturn&=IOREAD(hInfile%,pBuffer&+r%*KReadChunk&,KReadChunk&)
  r%=r%+1
  BUSY GEN$(r%,6)
UNTIL IOreturn&<>KReadChunk&
BUSY OFF
IOCLOSE(hInfile%)
RETURN pBuffer&
ENDP

PROC Message:(aMsg$)
gIPRINT aMsg$,KBusyTopRight%
ENDP
```

dpNote 0019 **Converting long text buffers between Unicode and Ascii**

21 Dec 2002

OPL for v6.0
Series 80

These two procedures are useful when you have read a text file into a binary buffer, for instance using the routine described in **dpNote 0018**, and wish to covert it to either Unicode or Ascii format.

For conversion from Unicode to Ascii, the usage is:

```
pBuffer&=ConvertUnicodeToAscii&:(pBuffer&,BufferSize&)
```

For conversion from Ascii to Unicode, the usage is

```
pBuffer&=ConvertAsciiToUnicode&:(pBuffer&,BufferSize&)
```

The procedures require the Buffer.opx and the Convert.opx.

Also take note that a real program should have out of memory error handling.

```
INCLUDE "Buffer.oxh"
INCLUDE "Convert.oxh"
INCLUDE "Const.oph"

PROC ConvertUnicodeToAscii&:(apSource&,aSourceSize&)
// this routine converts Unicode text in a binary buffer apSource& to
// a buffer with the same text in Ascii. The size of the resulting
// buffer is half that of the original buffer
LOCAL SourceSize&,SourceLength&,TargetSize&,TargetLength&,pTarget&
LOCAL ChunkLength&,pIn&,pOut&
// get size of source buffer and calculate length
// using LENALLOC(apSource&) will give word alignment error
SourceSize&=aSourceSize&
SourceLength&=SourceSize&/KUnicodeFactor%
// the size of the target buffer is half of that of the source buffer
TargetLength&=SourceLength&
TargetSize&=TargetLength&
// allocate the target buffer
pTarget&=ALLOC(TargetSize&)
IF pTarget&=0 :RAISE KErrNoMemory% :ENDIF
// the Convert opx only allows to convert 255 characters at a time
ChunkLength&=KMaxStringLen%
// initialise pointers
pIn&=apSource&
pOut&=pTarget&
// convert 255 characters at a time
WHILE (pIn&-apSource&)<=(SourceSize&-KMaxStringLen%*KUnicodeFactor%)
  CvFromUnicode:(pOut&,ChunkLength&,BufferAsString$:(pIn&,ChunkLength&))
  // the source buffer pointer is offset 2 times more than the target
  pIn&=pIn&+ChunkLength&*KUnicodeFactor%
  pOut&=pOut&+ChunkLength&
  // check if the pointer is closer than 255 chars to the end of buffer
```

```
ENDWH
// handle the remaining chunk
ChunkLength&=(SourceSize&-(pIn&-apSource&))/KUnicodeFactor%
CVFromUnicode:(pOut&,ChunkLength&,BufferAsString$:(pIn&,ChunkLength&))
// free up the source buffer
FREEALLOC apSource&
// return pointer to target buffer
RETURN pTarget&
ENDP

PROC ConvertAsciiToUnicode&:(apSource&,aSourceSize&)
// this routine converts Ascii text in a buffer apSource& to
// a buffer with the same text in Unicode. The size of the resulting
// buffer is double that of the original buffer
// i.e. TargetSize&=aSourceSize&*KUnicodeFactor%
LOCAL SourceSize&,SourceLength&,TargetSize&,TargetLength&,pTarget&
LOCAL ChunkLength&,pIn&,pOut&
// get size of source buffer and calculate length
SourceSize&=aSourceSize&
SourceLength&=aSourceSize&
// the size of the target buffer is double that of the source buffer
TargetLength&=SourceLength&
TargetSize&=TargetLength&*KUnicodeFactor%
// allocate the target buffer
pTarget&=ALLOC(TargetSize&)
IF pTarget&=0 :RAISE KErrNoMemory% :ENDIF
// the convert opx only allows to conversion of 255 characters at a time
ChunkLength&=KMaxStringLen%
// initialise pointers
pIn&=apSource&
pOut&=pTarget&
// convert 255 characters at a time
WHILE (pIn&-apSource&)<=(SourceSize&-KMaxStringLen%)
  BufferFromString&:(pOut&,ChunkLength&,CvUnicode$:(pIn&,ChunkLength&))
  // the target buffer pointer is offset 2 times more than the source
  pIn&=pIn&+ChunkLength&
  pOut&=pOut&+ChunkLength&*KUnicodeFactor%
  // check if the pointer is closer than 255 chars to the end of buffer
ENDWH
// handle the remaining chunk
ChunkLength&=(SourceSize&-(pIn&-apSource&))
BufferFromString&:(pOut&,ChunkLength&,CVUnicode$:(pIn&,ChunkLength&))
// free up the source buffer
FREEALLOC apSource&
// return pointer to target buffer
RETURN pTarget&
ENDP
```

Convention: We always use the word 'length' or 'len' to refer to the length of a string in characters. We always use the word 'size' to refer to a the size of a string in bytes. In ER5 days, there was no apparent difference, so in the documentation of the Buffer.opx, which survive to ER6 days, sometimes the two terms are confused.

dpNote 0020 **LOC function which gives correct answer for control and Unicode characters**

OPL for v6.0 Series 80

The `LOC(aString$,aChar$)` function in OPL does not work correctly in v6/S80 for control and Unicode characters.

As a workaround, the following procedure can be used. It requires the Buffer OPX.

```
INCLUDE "Buffer.oxh"
INCLUDE "Const.oph"

PROC Loc%:(aString$,aChar$)
LOCAL String$(KMaxStringLen%),Location%
String$=aString$
Location%=BufferFind&:(ADDR(String$)+KUnicodeHeader%,LEN(String$),aChar$,0)+1
RETURN Location%
ENDP
```

Thanks to Kevin Millican for this idea.

dpNote 0027 **Launching an application from OPL and wait until it finishes before returning**

21 June 2001

OPL for v5.0 Eikon

The following demonstrates how to launch a Word file then 'log on' to the returned thread and wait until it has been ended by the user before continuing. The code makes use of asynchronous event handling.

```
INCLUDE "System.oxh"
```

```
INCLUDE "Const.oph"

PROC Main:
RunEndReturn:("C:\Documents\CloseMe")
ENDP

PROC RunEndReturn:(aFilename$)
LOCAL File$(KMaxStringLen%),Ev&(16),EventStatus%
LOCAL ThreadID&,ThreadStatus& // For our Word thread
IF EXIST(aFilename$)
  TRAP DELETE aFilename$
ENDIF
// Start Word and create the file.
// The OPL application is pushed to the background...
ThreadID&=RunApp&:("Word",aFilename$,"",1)
// ...but we keep running, so keep an eye on the thread.
LogOnToThread:(ThreadID&,ThreadStatus&)
WHILE KTrue%
  // Queue an async event read.
  GETEVENTA32 EventStatus%,Ev&()
  PRINT "IOWAITing..."
  IOWAIT
  // Check for thread completion
  IF ThreadStatus&<>KStatusPending32&
    // We may still be in background,
    // so let someone know what's happened.
    BEEP 3,300
    PRINT "Word has finished."
    BREAK
  // Check for other events
  ELSEIF EventStatus%<>KErrFilePending%
    // Look at the event type...
    IF Ev&(KEvType%)=KEvFocusGained& // foreground
      PRINT "We're in the foreground."
      BREAK
    ELSE
      PRINT "Other event",HEX$(Ev&(KEvType%)),"ignored."
    ENDIF
  ENDIF
ENDWH
PRINT "Done."
GET
ENDP
```

From Symbian Knowledgebase FAQ-0412

dpNote 0028 **Asynchronous event loop with inactivity timer**

1 June 2003 (updated with comments on 1 Oct 2003)

OPL for v5.0 Eikon

and

OPL for v6.0 Series 80

The following is an application template which works both for v5/Eikon (in which we have used this structure in several applications) and v6/S80 (in which we have used it for one application). It has been tested on:

- Psion netBook (v5/Eikon)
- WINS for v5/Eikon
- WINS for v6/S80
- Nokia 9210 (v6/S80)

When running in v5/Eikon, it works as expected and no stray signals are ever generated.

When running in v6/S80, lots of stray signals are generated. Furthermore, if not two `IOSIGNAL` are added to the stray signal handling, the application will hang after a while. Therefore, we had to add a `KAddSomeExtraIoSignalB%` which should be set to `KFalse%` in v5/Eikon and to `KTrue%` in v6/S80.

We have not tried this in v6.1/S60, v7/S60, v7/UIQ, v7/S80, nor v7/S90 yet.

We think that asynchronous event handling is a very important part of OPL. It has been indicated before that v6/S80 is quite flaky in this area but we hope to be able to identify where the problems are in order to make it better going forward. In particular, there is reportedly a buggy timer.

```
// Asynchronous Event Handling
// Getevent loop with inactivity timer
// - can be used for 'screensavers' etc
// 19/6/2003 DP Pte Ltd
// www.dp.com.my

INCLUDE "Const.oph"
```

```
INCLUDE "System.oxh"

// select platform before compilation
rem CONST KSymbianOsVersion$="v5/Eikon"
rem CONST KSymbianOsVersion$="v6/S60"
CONST KSymbianOsVersion$="v6/S80"
rem CONST KSymbianOsVersion$="v7/UIQ"
rem CONST KSymbianOsVersion$="v7/S60"
rem CONST KSymbianOsVersion$="v7/S80"
rem CONST KSymbianOsVersion$="v7/S90"

CONST KTimeOut%=5 // seconds
CONST KTimerStart&=0 // reset timer

// needs to be set KTrue% for v6/S80 and to KFalse% for v5/Eikon
// otherwise application will hang
// but should not be needed were OPL for v6/S80 bugfree
CONST KAddSomeExtraIoSignalB%=KTrue%

PROC Main:
LOCAL t$(KMaxStringLen%),a$(KMaxStringLen%),b$(KMaxStringLen%)
t$="Eventloop with non-activity timer"
a$="This is a template for application"+NewLine$:+"development"
b$=NewLine$:+"This machine is: "+GetMachineName$:
InfoDialogue:(t$,a$,b$)
EventLoopA:
ENDP

PROC EventLoopA:
LOCAL ev&(16),EventStatus%
LOCAL TimeOut&,hTimer%,TimerStatus%,TimerReturn%
LOCAL StraySignalCount%
LOCAL ForeGroundB%
// program starts in foreground
ForeGroundB%=KTrue%
// reset stray signal counter
StraySignalCount%=0
// create a timer
IOOPEN(hTimer%,KIOTimer$,KIOModeDeviceOnly%)
// ensure at least one loop
ev&(1)=0
// start getevent loop
WHILE ev&(1)<>KKeyEsc%
  // set async
  GETEVENTA32 EventStatus%,ev&()
  // set timer only if in foreground
  IF ForeGroundB%
    TimeOut&=KTimeOut%*10
    TimerReturn%=IOC(hTimer%,1,TimerStatus%,TimeOut&,#KTimerStart&)
  ENDIF
  // wait for something to happen
  IOWAIT
  IF EventStatus%<>KStatusPending% // event
    // cancel timer only if in foreground, i.e. if it has been
    // initialised earlier
    IF ForeGroundB%
      IOCANCEL(hTimer%)
      IOWAITSTAT hTimer%
    ENDIF
    // handle input events
    IF ev&(KEvType%)=KEvFocusLost& // program moved to background
      Message:("Disappearing...")
      // option to have a HotKeyHandler: procedure here
      ForeGroundB%=KFalse%
    ELSEIF ev&(KEvType%)=KEvFocusGained& // program moved to foreground
      Message:("Reappearing...")
      ForeGroundB%=KTrue%
    ELSEIF ev&(KEvType%)=KEvDateChanged&
      Message:("Tempus fugit...")
    ELSE // data input event
      IF (ev&(KEvType%)<>KEvKeyDown& AND ev&(KEvType%)<>KEvKeyUp&)
        IF (ev&(KEvType%)<>KEvPtr&) AND (ev&(KEvType%)<>KEvPtrEnter&) AND (ev&
(KEvType%)<>KEvPtrExit&)
          // replaces (ev&(KEvType%) AND KEvNotKeyMask&)=0
          EventKeyProcess:(ev&(KEvType%),ev&(KEvMod%),ev&(KEvScan%))
```

```
        ELSE
          EventPenProcess:(ev&(KEvWinID%),ev&(KEvPtrType%),ev&(KEvPtrX%),ev&(KEvPtrY%))
        ENDIF
      ENDIF
    ENDIF
  ELSEIF TimerStatus%<>KStatusPending% // time out
    GETEVENTC(EventStatus%)
    TimeOutProcess:
  ELSE // stray signal
    Message:("Stray signal count = "+GEN$(StraySignalCount%,5))
    StraySignalCount%=StraySignalCount%+1
    GETEVENTC(EventStatus%)
    IF ForeGroundB%
      IOCANCEL(hTimer%)
      IOWAITSTAT hTimer%
    ENDIF
    IF KAddSomeExtraIoSignalB%
      IOSIGNAL :IOSIGNAL // this one is crazee!!!
    ENDIF
  ENDIF // end of async status checks
ENDWH
// put stray signals back again
WHILE StraySignalCount%>0
  IOSIGNAL
  StraySignalCount%=StraySignalCount%-1
ENDWH
Info:("Finished")
ENDP

PROC EventKeyProcess:(aKeyType&,aKeyMod&,aKeyScan&)
Message:("Key "+HEX$(aKeyType&)+" "+HEX$(aKeyMod&))
ENDP

PROC EventPenProcess:(aPtrWID&,aPtrType&,aPtrX&,aPtrY&)
Message:("Pen "+GEN$(aPtrWID&,2)+" "+GEN$(aPtrX&,3)+" "+GEN$(aPtrY&,3))
ENDP

PROC TimeOutProcess:
Message:("Time out!")
ENDP

PROC ___standard_stuff: :ENDP

PROC Info:(aText$)
InfoDialogue:(aText$,"","")
ENDP

PROC InfoDialogue:(aTitle$,aLine1$,aLine2$)
LOCAL Title$(KMaxStringLen%),Text$(KMaxStringLen%)
LOCAL choice%
IF aTitle$="Error"
  dINIT "Error"
  dTEXT "",ERRX$,KdTextCentre%
  dTEXT "",ERR$(ERR),KdTextCentre%
  GOTO LBL_Dialog::
ELSEIF aTitle$=""
  Title$="Note"
ELSE
  Title$=aTitle$
ENDIF
Text$=aLine1$
IF aLine2$<>""
  Text$=Text$+NewLine$:+aLine2$
ENDIF
dINIT Title$
IF Text$<>""
  dTEXT "",Text$
ENDIF
LBL_Dialog::
dBUTTONS "Close",KKeyEnter%
LOCK ON :choice%=DIALOG :LOCK OFF
ENDP

PROC NewLine$:
IF KSymbianOsVersion$="v5/Eikon"
```

```
  RETURN CHR$($0a) // ASCII
ELSE
  RETURN CHR$(KLineFeed&) // Unicode
ENDIF
ENDP

PROC GetMachineType&:
LOCAL value&,return&
value&=0
return&=SyGetHAL&:(KSyMachineUID&,value&)
RETURN value&
ENDP

PROC GetMachineName$:
LOCAL value&
IF KSymbianOsVersion$="v5/Eikon"
  RETURN MachineName$:
ENDIF
value&=GetMachineType&:
IF value&=&10005f62 // KSyMachineUid_Win32Emulator&
  RETURN "Win32 ER6 Emulator"
ELSEIF value&=&10005e33 // KSyMachineUid_Linda&
  RETURN "Nokia 9200 Series"
ELSEIF value&=&1000118a // KSyMachineUid_Series5mx&
  RETURN "Psion Series 5mx"
ELSE
  RETURN "unknown"
ENDIF
ENDP
```

dpNote 0029

1 June 2003

OPL for v6.1 Series 60

and

OPL for v6.0 Series 80

Key event codes for Series 60 phones

Compared to OPL for v6.0 Series 80, your `Const.oph` for Series 60 development will need some additional key event codes. A few existing ones also deserve some additional comments.

```
// Key constants (for 32-bit keywords like GETEVENT32)
CONST K32BitKeywordLimit&=&f000

CONST KKeyPageLeft&=&f802     // WINS only
CONST KKeyPageRight&=&f803    // WINS only
CONST KKeyPageUp&=&f804       // WINS only
CONST KKeyPageDown&=&f805     // WINS only
CONST KKeyEdit&=&f806         // Series 60 only
CONST KKeyLeftArrow&=&f807    // Series 80 and 60
CONST KKeyRightArrow&=&f808   // Series 80 and 60
CONST KKeyUpArrow&=&f809      // Series 80 and 60
CONST KKeyDownArrow&=&f80a    // Series 80 and 60
// For the command button array
CONST KKeyCBA1&=&f842         // Series 80 and 60 (Left CBA)
CONST KKeyCBA2&=&f843         // Series 80 and 60 (Right CBA)
CONST KKeyCBA3&=&f844         // Series 80 and WINS
CONST KKeyCBA4&=&f845         // Series 80 and 60 (Select CBA)
// Special keys
CONST KKeySidebarMenu&=&f700  // WINS configured as in dpNote 0010 only
CONST KKeyZoomIn&=&f703       // WINS only
CONST KKeyZoomOut&=&f704      // WINS only
CONST KKeyMenu&=&f836         // Series 80 and WINS
CONST KKeyHelp&=&f83a         // Series 80 and WINS
CONST KKeyApplications&=&f852 // Series 60 only
CONST KKeySend&=&f862         // Series 60 only
CONST KKeyEndSend&=&f863      // Series 60 only
CONST KKeyBrightness&=&f864   // Series 80 only

// the Clear-key or [C]-key is actually Unicode Backspace
CONST KKeyBackspace&=&0008    // Series 80 and 60 (Clear)
CONST KKeyClear&=KKeyBackspace&

// The numeric keys including '*' and '#' follow Unicode values just like
// in Series 80 phones
CONST KKeyHash&=&0023
CONST KKeyNumberSign&=KKeyHash&
CONST KKeyStar&=&002a
CONST KKeyAsterisk&=KKeyStar&
CONST KKey_0&=&0030
CONST KKey_1&=&0031
```

```
CONST KKey_2&=&0032
CONST KKey_3&=&0033
CONST KKey_4&=&0034
CONST KKey_5&=&0035
CONST KKey_6&=&0036
CONST KKey_7&=&0037
CONST KKey_8&=&0038
CONST KKey_9&=&0039
```

Take note of the consistent use of `KKeyCBA4&` as the Select-key for both Series 60 and Series 80 usage and style guide.

Disclaimer: At the time of writing, OPL for Series 60 has reached v0.25 Alpha. With the continued development of OPL for Series 60 phones additions and amendments will likely be made to this dpNote. Most likely, for instance, in the future a Series 60 WINS console will be able to simulate additional Series 60 keypresses.

Links to other OPL developer resources

http://www.opl32.com/Menu_Fiches.htm - even if you cannot read French, this site offers a number of very good OPL for Symbian OS v5 related notes. It is not updated very often though.

http://www.allaboutopl.com/wiki/OPLWikiHome - this site has the purpose of being the central Documents and Reference site for OPL, in particular for Symbian OS v6.0 and later. It is a Wiki site, meaning that anyone can register and contribute to the OPL projects.