Extended OPL Calls

Extended OPL CallsExtended OPL
Calls

Introduction This document is an attempt to list the may possible options available to the OPL-Programmer, which are normally only to be found in the SIBO 'C'-Programming Guides.

The various tips and tricks are illustrated with examples in Standard OPL.

There is no particular order to the points listed. The information can be freely distributed, but is Psion GmbH. Thanks to David Wood at Psion PLC for helping me with these routines over the past few months.

Neither the Author, nor Psion GmbH, takes any responsibility for and damage or loss of data which occurs as result of using information in this document. The information is subject to change without notice.

Comments are shown in italics.

Use of CALL and OS

CALLs and OSs take a particular format. This is briefly explained in the OPL Programming Manual, but is repeated here for reference.

Using the 'Service' and 'Interrupt' reference in the 'C'-Programming Guide, it is possible to access the specific operating system call desired.

Format for Call:

```
ret%=CALL($xxyy,...,) xx = Service, yy = Interrupt
```

Format for OS:

```
LOCAL ax%,bx%,cx%,dx%,si%,di% keep these together
LOCAL flags%
ax%=$xxyy xx = Service, yy = AL (parameter)
flags%=OS($zz,addr(ax%)) zz = Interrupt
```

If an error occurs when using OS, (flags% AND 1) will be true, the error code is given by ax% AND $ff00.

Starting another process

Here we start another process, and specify the name of the file to be opened. The command line starts with 'C' for create, or 'O' for open. The filename is stored in f$. The 'Record' at the beginning of the command line in this example is the name of the icon under which the filename will appear in the System Screen. If this is not a valid (ie. installed) icon, then the filename will appear under RunImg.

eg. Recorder with a filename

```
PROC Record:(f$)
   LOCAL cmdl$(128),helpnm$(128),hpid%,ret%     cmdl$="CRecord"+chr$(0)+".WVE"+"
"+chr$(0)+f$+chr$(0) command line
   helpnm$="rom::record.app"+chr$(0)
ret%=call($0187,addr(helpnm$)+1+addr(cmdl$),0,0,addr(hpid%)
   call($0688,pid%) ProcResume
ENDP
```

Get Process ID

Each process has an ID number. Here we read the ID number for a specific process.

eg. Get the process ID for the system shell. pid% is defined globally to hold this variable.

```
   name$="sys$shll.*"
   pid%=call($0188,addr(name$)+1) ProcIdByName
```

Getting the Process ID for the current process

This short piece of code reads the ID for the current process.

```
     LOCAL pid%
     pid%=CALL($0088)
```

Setting the priority of the process

Once we have the ID for a process, we can change the priority of this process.

```
     LOCAL ax%,bx%,cx%,dx%,si%,di% keep these together     LOCAL flags% NB: pid% must be defined to be the process id

     bx%=pid%
     ax%=$0398

     flags%=OS($88,addr(ax%)) ProcSetPriority
```

Changing the position of a process

We can also change the position of this process, ie. whether it is in foreground or background.

```
     CALL($998d,pos%,pid%)

     For the current process, pid% can be set to zero.     For foreground, use pos%=0, for background, pos%=100.
```

Language Code

Programs written by Psion are in the majority Multi-lingual. This means that if they run on an English machine, they run in English, on a German machine they run in German, and so on.

Each language is assigned a number, which is used to recognise the machine in use.

This example returns a string containing the number of the code for the resource file.

```
PROC Lang$:
     LOCAL ax%,bx%,cx%,dx%,si%,di% Keep these variables together
     LOCAL flags%,a$(2)
```

```
     ax%=$1B00 GetLangData

     flags%=OS($008B,ADDR(ax%)) General Services

     IF flags% AND 1
          RETURN("01") an error occured
     ELSE
          a$=NUM$(ax%,2)          IF LEN(a$)<2 :a$="0"+a$ :ENDIF makes two digits for
filenames
     ENDIF
  ENDP

  The numbers currently in use are:
        1   English
        2   French
        3   German
        4   Spanish
        5   Italian
        6   Swedish
        7   Danish
        8   Norwegen
        9   Finish
        10  American
        11  Swiss French
        12  Swiss German
        13  Portuguese
        14  Turkish
        15  Icelandic
        16  Russian
        17  Hungarian
        18  Dutch
        19  Belgian Flemish
        20  Australian
        21  New Zealand
        22  Austrian
        23  Belgian French
```

```
     Simulating a key press
    There is a technique for the Series3a to simulate a keypress in another application.  This
method only works for Object-oriented applicaitions.

  The following must be globally defined:
     GLOBAL k%,m% keep these two together

  pid% is the process id, k% is the keycode, m% is the modifier
CALL($0483,pid%,$31,0,addr(k%)) MessSend



  Capturing a key in background
    We can also when a particular key is pressed, even if the process requiring this key is not
current.

     call($c58d,26,$404) wCaptureKey
     This example captures key 26, ie. Ctrl-Z.



  Capturing the off key

  The following parameters will capture the 'OFF' key.

     call($c58d,$2003,$e08)

     keya(kstat%,k%(1) queues for a key press
       The value returned in kstat% will NOT be -46 if a key is pressed, and for the off
key,
     k%(1) will be $2003.

    Reading an environment variable
    Environment variables are used to store data which is used by all applications.  Space for
these variables is limited, and should normally not be used by OPL applications.

  PROC EnvGet:
     LOCAL ax%,bx%,cx%,dx%,si%,di% keep these together
```

```
     LOCAL flags%

     LOCAL env$(6),penv%,lenenv%
     LOCAL buff$(255),pbuff%,lenbuff%

     env$="$WP_PW" name of the variable to search for, eg. $WP_PW
     penv%=ADDR(env$+1)
     lenenv%=LEN(env$)

     pbuff%=ADDR(buff$)+1

     ax%=$2100
     bx%=0
     di%=penv%
     dx%=lenenv%
     si%=pbuff%

     flags%=OS($008b,ADDR(ax%)) GenEnvBufferGet

     IF flags% AND 1
          An error has occured, error code: ax% OR $ff00
     ELSE
          lenbuff%=ax%
          POKEB ADDR(buff$),lenbuff% Insert leading count
     ENDIF
  ENDP
```

  Reading the user details
  User details are stored in environment variable $WS_PW. Bytes 4, 8, 12, and 16 contain the length (in bytes) of each of the four lines.
  The first line begins at byte 19.

  Reading the current printer driver The printer destination is stored in environment variable P$D. Zero for parallel, one for serial, and two for file. The printer driver is stored in environment variable P$M. This is generally a filename. If printing to a file, the name is stored in P$F.

```
  Asynchronous WVE Playing

This procedure will play a .WVE file asynchronously.

PROC Play:(inname$,ticks%,vol%)
   LOCAL name$(128),pstat%
   name$=inname$+CHR$(0)
   CALL($1E86,UADD(ADDR(name$),1),ticks%,vol%,0,pstat%)
   IOWAITSTAT pstat%
ENDP



Cancelling Asynchronous Wave Playing
PROC Playc:
   CALL($2086)
ENDP

  Window Server os-calls

This is a list of the Window Server os-calls, given here for reference purposes only.

gSetOpenAddress(bx,cx,dx)   $5f8d
wCancelCaptureKey(bx,cl,ch) $c68d
wCancelSystemModal(bx) $c88d
wCaptureKey(bx,cl,ch)  $c58d
wClientInfo(bx)     $8c8d
wClientPosition(bx,cx) $998d
wDisableLeaves(dx)  $0dd6
wDisablePauseKey()  $ce8d
wDrawButton(bx,cx,dx)  $608d
wEnablePauseKey()   $4d8d
wEndCompute()       $8b8d
wGetProcessList(si) $d98d
wSendCommand(bx,cx,dx) $da8d
wStartCompute()     $8a8d
wsUpdate(bx)        $528d
wSystemModal(bx)    $c78d
```

```
    Power Status

  This code will return TRUE if the mains is plugged in, and FALSCH if not.

  PROC mainsin%:
     LOCAL esup%(3)
     CALL($118e,addr(esup%(1)))   HwGetSupplyStatus
     RETURN esup%(3)
  ENDP

  Auto-off settings
    This code will return TRUE if the Series3a is NOT going to turn off, eg. because the
external power is plugged in.

  PROC notifm%:
     IF (PEEKB(PEEKW($18)+13) and $f <2
          return 0 This is not a 3a     ELSEIF (CALL($388b) AND $FF) <> 1
GenGetAutoMains
          return 0
     ELSE
          return(mainsin%:)
     ENDIF
  ENDP
    Now we can read the number of seconds to go before the Auto-off turns the machine off.
Note: In OPL  this will only work if we first call GenMarkNotActive and wEndCompute.

     ie.  CALL($138b) GenMarkNotActive
          CALL($8b8d) wEndCompute

  PROC timelft%:
     LOCAL gooff%,timeoff% keep these together
     IF notifm%:
          RETURN -1
     ENDIF
     CALL($078b,0,4,0,$40a,ADDR(gooff%)) GenGetOsData
     IF gooff%<0
```

```
          RETURN -1
     ELSE
          RETURN(timeoff%)
     ENDIF
  ENDP
```

Asynchronous reading

It is possible to read keys and timers asynchronously. Firstly, a key:

```
  PROC quekey:
     keya(kstat%,k%(1))
  ENDP
```

kstat% and k%(2) are globally defined. Kstat% will by -46 if no key was pressed. If the off key is pressed (and has been captured), then k%(1) will be $2003, otherwise the values in k%() will be similar to those returned using GETEVENT.

Now the timer:

Firstly we need to globally define timh%, timstat%, and then we open the channel to the timer function.

```
     IOOPEN(timh%,"TIM:",-1)
```

Now we can queue the timer.

```
  PROC quetim:
     LOCAL t&,t%
     t%=2 This is the number of seconds before the timer should expire
     t&=int(t%*10)
     ioa(timh%,1,timstat%,t&,#0)
  ENDP
```

Further Reading

This has been only a brief guide to some of the functions available on the Series3a. For more information, see the SIBO 'C'-Programming Guide.

Revision #2

Created Thu, Jan 24, 2019 12:25 PM by Alex

Updated Thu, Jan 24, 2019 12:25 PM by Alex