```
// AKUA Protos OneFile
#include      "yLibCfg.h"

#include      "Reaper.h"

#include      "yAgtINIT.h"
#include      "yAgtTRAP.h"
#include      "yDraw.h"
#include      "yFixedMath.h"
#include      "yGlobal.h"
#include      "yString.h"

#include      <Appearance.h>



//
// For PPC Code resource
#if             ISAPPC
ProcInfoType      __procinfo = bpiObjAgt;
#endif   //   ISAPPC


// AKUA Statics Beg
oaVal main(oaObj initBlkObj);
static SysProc tpLaunchX(LaunchPBPtr pb, OSErr * err);
#if             ISAPPC
static pascal OSErr xaLaunchPPC(LaunchPBPtr pb);
#else    //   ISAPPC
static pascal asm OSErr xaLaunch(void);
#endif   //   ISAPPC
static SysProc tpGet1ResourceX(ident kind, short resNum, Handle * retHdl);
#if             ISAPPC
static pascal Handle xaGet1ResourcePPC(ResType kind, short resNum);
#else    //   ISAPPC
static pascal asm Handle xaGet1Resource(ResType kind, short resID);
#endif   //   ISAPPC
static Handle tpNewHandleX(word tw, Size amt);
#if             ISAPPC
static pascal Handle xaNewHandlePPC(word tw, Size amt);
static pascal Handle xaNewHandleSmartPPC(word tw, Size amt);
#else    //   ISAPPC
static pascal asm void xaNewHandle(void);
static pascal asm void xaNewHandleSmart(void);
#endif   //   ISAPPC
static Ptr tpNewPtrX(word tw, Size amt);
#if       ISAPPC
static pascal Ptr xaNewPtrPPC(word tw, Size amt);
#else
static pascal asm void xaNewPtr(void);
#endif   //   ISAPPC
static SysProc tpDisposePtrX(word tw, Ptr p);
#if             ISAPPC
static pascal void xaDisposePtrPPC(word tw, Ptr p);
#else    //   ISAPPC
static pascal asm void xaDisposePtr(void);
#endif   //   ISAPPC
static SysProc tpMaxApplZoneX(void);
#if             ISAPPC
static pascal void xaMaxApplZonePPC(void);
#else    //   ISAPPC
static pascal asm void xaMaxApplZone(void);
```

```
#endif   //   ISAPPC
static PicHandle tpOpenPictureX(rect box);
#if              ISAPPC
static pascal PicHandle xaOpenPicturePPC(rect box);
#else    //   ISAPPC
static pascal asm void xaOpenPicture(void);
#endif   //   ISAPPC
yError tpInit(initBlk init);
SysProc osTrapSwap(word trapWord, void * newTrapAdr);
// AKUA Statics End



enum TrapReplacement

      ktrLaunch,
      ktrGet1Resource,
      ktrMaxApplZone,

      ktrOpenPicture,        // Only set at MaxApplZone
      ktrNewHandle,
//    ktrSetHandleSize,
      ktrNewPtr,
      ktrDisposePtr,

      ktrCnt
 ;


// Our globals
ReaperGlo     grp =

      igsReaper,
      kvrReaperCurrent,
      0,
      NULL
 ;

MonkeyFlag   theMonkey;
SysProc       gatTraps[ktrCnt];
// End globals




oaVal main(oaObj initBlkObj)

      EnterCodeRsrc();

      initBlk        init = (initBlk)initBlkObj;

      // bkkEternal - since we stay resident, load and detach 'PREF' resource
      init->flags = bkkEternal;

      // Global Initialization
      theMonkey = bEvMonkeyLives | (init->monkey & (bEvMonkeyPowered | bEvMonkeyHasColourQD |
bEvMonkeyUseColourQD));

      grp.prefs = (BADAPP)init->prefs;
```

```
    // Set up traps
    yError err = tpInit(init);          // Init Gestalt Entries

    // Show our icon, get prefs
    CallObjAgt(init->kickAgt, init);

deadlyError:
    LeaveCodeRsrc();

    return   err;




static SysProc tpLaunchX(LaunchPBPtr pb, OSErr * err)

 /* ____________________________________________________________________
    tp          LaunchX

    In
    Out

    Effect   Grab events, check 'em out, and pass 'em on…
             (Keep track of idle for AutoOK function and
              pop-up utility menus when the user has chosen them).

    Const
    Errors
    Flags
    Global
    Rsrc

    Version 001
    Notes
    History
    001 GOD 12.07.95 wdSelect action moved here
              Action now includes 2 secs or more of
              events not being processed.
 ____________________________________________________________________ */
    EnterCodeRsrc();

    SysProc trap = gatTraps[ktrLaunch];

    mcObjSet(&grp, brpIsLaunching);

#if              ISAPPC
    // DebugStr("\pLaunch");
    *err = CallUniversalProc(trap, bpiLaunch, pb);
#else    //   ISAPPC
    *err = (*(trapLaunchProc)trap)(pb);
#endif   //   ISAPPC

    mcObjClr(&grp, brpIsLaunching);

    LeaveCodeRsrc();

    // return     trap;
    return          NULL;
```

```
#if             ISAPPC

static SysProcDef xaLaunch = BUILD_ROUTINE_DESCRIPTOR(bpiLaunch, xaLaunchPPC);

static pascal OSErr xaLaunchPPC(LaunchPBPtr pb)

    OSErr          err;

    if (SysProc p = tpLaunchX(pb, &err))
        err = CallUniversalProc(p, bpiLaunch, pb);

    return   err;


#else    //   ISAPPC

static pascal asm OSErr xaLaunch(void)

    // Parms are in registers!!! Careful
    clr.l          -(sp)                                  // Space for real trap
    movem.l        a0-a1/d0-d2, -(sp)

    pea             4 + 20 + 4(sp)                        // Space, Regs, RTS Adress (the value of this address is also in d0)
    move.l          a1, -(sp)                             // Event Record *

    jsr             tpLaunchX
    addq.w          #8, sp                                 // Kill parms

    move.l          a0, 20(sp)                            // New rts
    movem.l         (sp)+, a0-a1/d0-d2

    move.w          4(sp), d0                              // Restore d0 from new return value
    tst.l          (sp)
    bne.s           @1
    addq.w          #6, sp                                // Kill NULL Trap & word return
@1: rts


#endif   //   ISAPPC



static SysProc tpGet1ResourceX(ident kind, short resNum, Handle * retHdl)

 /* ____________________________________________________________
    tp          Get1ResourceX

    In
    Out

    Effect   Grab events, check 'em out, and pass 'em on…
             (Keep track of idle for AutoOK function and
              pop-up utility menus when the user has chosen them).

    Const
    Errors
    Flags
    Global
    Rsrc
```

```
    Version 001
    Notes
    History
    001 GOD 12.07.95 wdSelect action moved here
                Action now includes 2 secs or more of
                events not being processed.
_____________________________________________________________ */
    EnterCodeRsrc();

    SysProc trap = gatTraps[ktrGet1Resource];

    if ((kind == ircSizeOfHeap) && mcObjTst(&grp, brpIsLaunching))

#if          ISAPPC
        if (Handle hdl = (Handle)CallUniversalProc(trap, bpiGet1Resource, kind, resNum))
#else   //   ISAPPC
        if (Handle hdl = (Handle)(*(tpGet1ResourceProc)trap)(kind, resNum))
#endif  //   ISAPPC

            *retHdl = hdl;
            trap = NULL;

            long      originalSize = *(long *)(*hdl + 2);

            if (!(originalSize & 0xFF))

                mmHdlPurgeOff(hdl);
                *(long *)(*hdl + 2) = rpFindHeapSize(*grp.prefs, CurResFile(), originalSize);



    LeaveCodeRsrc();

    return   trap;



#if          ISAPPC

static SysProcDef xaGet1Resource = BUILD_ROUTINE_DESCRIPTOR(bpiGet1Resource, xaGet1ResourcePPC);

static pascal Handle xaGet1ResourcePPC(ResType kind, short resNum)

    Handle                      h;

    if (SysProc p = tpGet1ResourceX(kind, resNum, &h))
        return   (Handle)CallUniversalProc(p, bpiGet1Resource, kind, resNum);

    return   h;


#else   //   ISAPPC

static pascal asm Handle xaGet1Resource(ResType kind, short resID)

//  180 GOD 20.03.96      Lots o' changes, including saving A1 around GetResource since ViseInstaller presumes this
//                        the dickheads!

    enum ParmSizes  parmsSize = sizeof(ResType) + sizeof(short) ;

    movem.l       d1-d2/a1, -(sp)
```

```
    subq.w      #4, sp                          // Space for return
    pea         (sp)                            // Pointer to it
    move.w      4 + 8 + 12(sp), -(sp)           // resID @ RetAdr + RetVal + Regs
    move.l      4 + 8 + 12 + 4(sp), -(sp)       // kind @ RetAdr + RetVal + Regs + resID + resID(parm)
    jsr         tpGet1ResourceX
    addq.w      #4, sp
    addq.w      #6, sp
    move.l      (sp)+, parmsSize + 4 + 12(sp);  // Parms + RetAdr + SavedRegs
    movem.l     (sp)+, d1-d2/a1

    // Return in a0 for MWC!
    move.l      a0, d0
    bne.s       @1
    move.l      (sp)+, a0           // Caller's address
    addq.w      #parmsSize, sp      // We be pascal
    move.w      kosResErr, d0       // Set r0
@1: jmp         (a0)


#endif  //  ISAPPC




static Handle tpNewHandleX(word tw, Size amt)

 /* ____________________________________________________________________
    tp Trap NewHandleX

    In
    Out
    Errors
    Effect  Replace known names with icon suites
    Const
    Global
    Rsrc
    Notes   EnterCodeRsrc() -- SetCurrentA4() must be done by 68K caller!!!
 ____________________________________________________________________ */
    Handle          h;
    OSErr           err;

    if(!(h = TempNewHandle(amt, &err)) || err)

        h = NULL;
        osMemErr = err;
        // Go for alternate zones - going for Sys may loop back to us!
//      h = NewHandleSys(amt);

    else if (mcFlagTst(tw, bosTrapIsClr))
        mmBlkClr(*h, amt);

    return  h;



// GLUE Code for tpNewHandle
#if         ISAPPC

static RoutineDescriptor xaNewHandle = BUILD_ROUTINE_DESCRIPTOR(bpiNewHandle, xaNewHandlePPC);
```

```
static pascal Handle xaNewHandlePPC(word tw, Size amt)

    Handle      h;


    if (!(h = (Handle)CallOSTrapUniversalProc(gatTraps[ktrNewHandle], bpiNewHandle, tw, amt)))
        if (!mcFlagTst(tw, bosTrapIsSys) && (osTheZone == osApplZone))
            h = tpNewHandleX(tw, amt);

    return  h;



static RoutineDescriptor xaNewHandleSmart = BUILD_ROUTINE_DESCRIPTOR(bpiNewHandle, xaNewHandleSmartPPC);

static pascal Handle xaNewHandleSmartPPC(word tw, Size amt)

    Handle      h       = NULL;
    bool        smart   = !mcFlagTst(tw, bosTrapIsSys) && (osTheZone == osApplZone);


    // From OpenPicture?
    if (smart)
        if (mcObjTst(&grp, brpInOpenPicture) || (amt > (osTheZone->zcbFree >> 2)))
            if (osTheZone->zcbFree < TempFreeMem()) // grp.tmpZone->zcbFree)
                h = tpNewHandleX(tw, amt);

    if (!h)
        if (!(h = (Handle)CallOSTrapUniversalProc(gatTraps[ktrNewHandle], bpiNewHandle, tw, amt)))
            if (smart)
                h = tpNewHandleX(tw, amt);

    return  h;


#else   //  ISAPPC

static pascal asm void xaNewHandle(void)      // Parameter in register d0, trapWord in d1

    movem.l     d1-d4/a1-a4, -(sp)

    move.l      d0, d3
    move.w      d1, d4

    // Access to our globals
    jsr         SetCurrentA4

//  _Debugger

    // In System Heap - Leave it alone
    btst        #bosTrapIsSysBit, d4
    bne         @7

    // Not App Zone - leave it alone
    move.l      kosApplZone, d0
    move.l      kosTheZone, d1
    cmp.l       d1, d0
    beq         @6

@7: // System or other zone - out of here!
    move.l      d3, d0
```

```
        move.l      gatTraps + (ktrNewHandle * 4), a0
        movem.l     (sp)+, d1-d4/a1-a4
        jmp         (a0)

        // Quick Check if that much is free
/*      move.l      kosTheZone, a0
        addq.w      #8, a0
        addq.w      #4, a0
        move.l      (a0), d0            // zcbFree
        sub.l       #10240, d0          // reserve for emergencies
        cmp.l       d3, d0
        blt         @4
*/
@6: // Call original NewHandle
        move.l      d3, d0
        // Restore d1 (trapWord)
        move.w      d4, d1
        move.l      gatTraps + (ktrNewHandle * 4), a0
        jsr         (a0)
        tst.w       d0
        beq         @2

@4: // _Debugger

        move.l      d3, -(sp)
        move.w      d4, -(sp)
        jsr         tpNewHandleX
        addq.w      #6, sp

@3: move.w          kosMemErr, d0

@2: movem.l         (sp)+, d1-d4/a1-a4   // Restores A4
@1: rts



static pascal asm void xaNewHandleSmart(void)     // Parameter in register d0, trapWord in d1

        movem.l     d1-d4/a1-a4, -(sp)

        move.l      d0, d3
        move.w      d1, d4

        // Access to our globals
        jsr         SetCurrentA4

        // In System Heap - Leave it alone
        btst        #bosTrapIsSysBit, d4
        bne         @7

        // Not App Zone - leave it alone
        move.l      kosApplZone, d0
        move.l      kosTheZone, d1
        cmp.l       d1, d0
        beq         @6

@7: // System or other zone - out of here!
        move.l      d3, d0
        move.l      gatTraps + (ktrNewHandle * 4), a0
        movem.l     (sp)+, d1-d4/a1-a4
        jmp         (a0)
```

```
@6: // How much do we have avail?
      move.l         kosTheZone, a0
      addq.w         #8, a0
      addq.w         #4, a0
      move.l         (a0), d0            // zcbFree
      move.l         d0, d1               // … to D1
      asr.l          #2, d0              // 1/4 to D0

      // Quick Check if that much is free
      cmp.l          d3, d0               // d0 (free >> 2) > d3 (amt) ?
      ble            @5

      // From OpenPicture()?
      move.w         grp.flags, d0
      btst           #brpInOpenPictureBit, d0
      beq            @0          // Go for the bigger zone

@5: // Check if more temp is available than our zone
/*
@5: move.l         grp.tmpZone, a0
      addq.w         #8, a0
      addq.w         #4, a0
      move.l         (a0), d0
*/
      move.l         d1, a3
      subq.w         #4, sp
      moveq          #0x18, d0                // _TempFreeMem
      move.w         d0, -(sp)
      _OSDispatch
      move.l         (sp)+, d0
      move.l         a3, d1
      cmp.l          d0, d1              // D1 (app) > D0 (tmp) ?
      bgt            @0                   // Skip it, app has more

      // Get some temp mem
      move.l         d3, -(sp)
      move.w         d4, -(sp)
      jsr            tpNewHandleX
      addq.w         #6, sp
      tst.w          kosMemErr
      beq            @3

      // Call original NewHandle
@0: move.l         d3, d0
      // Restore d1 (trapWord)
      move.w         d4, d1
      move.l         gatTraps + (ktrNewHandle * 4), a0
      jsr            (a0)
      tst.w          d0
      beq            @2

@4: // _Debugger

      move.l         d3, -(sp)
      move.w         d4, -(sp)
      jsr            tpNewHandleX
      addq.w         #6, sp

@3: move.w         kosMemErr, d0

@2: movem.l        (sp)+, d1-d4/a1-a4   // Restores A4
```

```
@1: rts

#endif   //   ISAPPC




static Ptr tpNewPtrX(word tw, Size amt)

 /* ____________________________________________________________
     tp Trap NewPtrX

     In
     Out
     Errors
     Effect   Replace known names with icon suites
     Const
     Global
     Rsrc
     Notes
 ____________________________________________________________ */
     EnterCodeRsrc();

     Ptr                  p;

#if             ISAPPC
     if (!(p = (Ptr)CallOSTrapUniversalProc(gatTraps[ktrNewPtr], bpiNewPtr, tw, amt)))
#else    //   ISAPPC
     if (!(p = (Ptr)(*(trapNewPtrProc)gatTraps[ktrNewPtr])(tw, amt)))
#endif   //   ISAPPC

          // Try system heap if we not already in System heap…
          if (!(tw & bosTrapIsSys))
#if             ISAPPC
               p = (Ptr)CallOSTrapUniversalProc(gatTraps[ktrNewPtr], bpiNewPtr, tw | bosTrapIsSys, amt);
#else    //   ISAPPC
               p = (Ptr)(*(trapNewPtrProc)gatTraps[ktrNewPtr])(tw | bosTrapIsSys, amt);
#endif   //   ISAPPC

          if (!p)

               Handle         h;

               if (h = tpNewHandleX(tw, amt + 12))

                    ident          * id;

                    mmHdlLock(h);
                    id = (ident *)*h;
                    *id++ = igsReaper;   // Mark as a pointer
                    *id++ = 'Ptr ';
                    *id++ = (lwrd)h;
                    p = (Ptr)id;



     LeaveCodeRsrc();
```

```
    return   p;


// GLUE Code for tpNewPtr
#if       ISAPPC

static RoutineDescriptor xaNewPtr = BUILD_ROUTINE_DESCRIPTOR(bpiNewPtr, xaNewPtrPPC);

static pascal Ptr xaNewPtrPPC(word tw, Size amt)

    return   tpNewPtrX(tw, amt);


#else

static pascal asm void xaNewPtr(void)     // Parameter in register d0

    movem.l       d1-d2/a1, -(sp)

    move.l        d0, -(sp)
    move.w        d1, -(sp)

    jsr           tpNewPtrX

    addq.w        #6, sp
    move.w        kosMemErr, d0
    movem.l       (sp)+, d1-d2/a1
    rts


#endif   //   ISAPPC




static SysProc tpDisposePtrX(word tw, Ptr p)

 /* ____________________________________________________________
    tp Trap DisposePtrX

    In
    Out
    Errors
    Effect   Replace known names with icon suites
    Const
    Global
    Rsrc
    Notes
 ____________________________________________________________ */
    EnterCodeRsrc();

    SysProc trap = gatTraps[ktrDisposePtr];

    // One of our pseudo pointers?
    if (!p)

        trap = NULL;
        osMemErr = 0;
```

```
    else if (   (*(ident *)(p - 8) == 'Ptr ')
                && (*(ident *)(p - 12) == igsReaper))

        trap = NULL;
        mmHdlDel(*(Handle *)(p - 4));


    LeaveCodeRsrc();

    return   trap;



// GLUE Code for tpDisposePtr
#if             ISAPPC

static RoutineDescriptor xaDisposePtr = BUILD_ROUTINE_DESCRIPTOR(bpiDisposePtr, xaDisposePtrPPC);

static pascal void xaDisposePtrPPC(word tw, Ptr p)

    if (SysProc trap = tpDisposePtrX(tw, p))
        CallOSTrapUniversalProc(trap, bpiDisposePtr, tw, p);


#else    //   ISAPPC

static pascal asm void xaDisposePtr(void)    // Parameter in register a0

    movem.l       d1-d2/a1, -(sp)

    move.l        a0, -(sp)
    move.w        d1, -(sp)                  // Trapword

    jsr           tpDisposePtrX

    addq.w        #2, sp                     // Trapword
    move.l        a0, d0

    move.l        (sp)+, a0
    movem.l       (sp)+, d1-d2/a1

    beq.s         @1
    move.l        d0, -(sp)                  // Real trap
@1: rts


#endif   //   ISAPPC




static PicHandle tpOpenPictureX(rect box)

 /* ____________________________________________________________
    tp Trap OpenPictureX

    In
    Out

    Effect   Replace NewHandle and NewPtr with our versions
```

```
                if the app is in our prefs resource and the
                traps have not been already zapped

    Const
    Errors
    Flags

    Global
    Rsrc

    Version 000
    Notes
    History
    000 GOD 10.04.96 Start
_____________________________________________________________ */
    EnterCodeRsrc();
    mcObjSet(&grp, brpInOpenPicture);

    PicHandle pic = CallTrapOpenPicture(gatTraps[ktrOpenPicture], box);

    mcObjClr(&grp, brpInOpenPicture);
    LeaveCodeRsrc();

    return   pic;



// GLUE Code for tpNewHandle
#if             ISAPPC

static RoutineDescriptor xaOpenPicture = BUILD_ROUTINE_DESCRIPTOR(bpiOpenPicture, xaOpenPicturePPC);

static pascal PicHandle xaOpenPicturePPC(rect box)

    return   tpOpenPictureX(box);


#else    //   ISAPPC

static pascal asm void xaOpenPicture(void)

    move.l        4(sp), a0              // Box parm
    move.l        (sp)+, (sp)            // Kill it
    movem.l       d0-d2/a1, -(sp)
    move.l        a0, -(sp)              // Box parm
//  _Debugger
    jsr           tpOpenPictureX
    addq.w        #4, sp                 // Kill box parm
    movem.l       (sp)+, d0-d2/a1
    move.l        a0, 4(sp)
    rts


#endif   //   ISAPPC




static SysProc tpMaxApplZoneX(void)
```

```
/* ____________________________________________________________________
    tp Trap MaxApplZoneX

    In
    Out

    Effect   Replace NewHandle and NewPtr with our versions
               if the app is in our prefs resource and the
               traps have not been already zapped

    Const
    Errors
    Flags

    Global
    Rsrc

    Version 000
    Notes
    History
    000 GOD 10.04.96 Start
____________________________________________________________________ */
    EnterCodeRsrc();

    bool      expansion     = FALSE;
    bool      smart         = FALSE;

//  mcObjClr(&grp, brpIsLaunching); // Can't be launching now?!?

    if (*osCurApName <= 31)

        ProcessSN               appSN;
        ProcessInfoRec          appInfo;
        FSObjName               appName;

        // Check if we are allowed to…
        appInfo.processInfoLength = sizeof(appInfo);
        appInfo.processName = appName;
        appInfo.processAppSpec = NULL;
        appSN.highLongOfPSN = 0;
        appSN.lowLongOfPSN = kCurrentProcess;
        GetProcessInformation(&appSN, &appInfo);

        if (appItem item = rpFindApp(*grp.prefs, appInfo.processSignature,
                                     appName, TRUE, NULL, NULL))

            expansion     = mcObjBool(item, bbaExpand);
            smart         = mcObjBool(item, bbaSmartHeap);


/*
        yError   err;

        if (1)
            grp.tmpZone = osTwitchZone;
        else if (Handle crap = TempNewHandle(3, &err))

            grp.tmpZone = HandleZone(crap);
            mmHdlDel(crap);

*/
    else
```

```
        // Reget trap info
        gatTraps[ktrOpenPicture]      = osTrapSwap(_OpenPicture, NULL);
        gatTraps[ktrNewHandle]         = osTrapSwap(_NewHandle, NULL);
        gatTraps[ktrNewPtr]             = osTrapSwap(_NewPtr, NULL);
        gatTraps[ktrDisposePtr]       = osTrapSwap(_DisposePtr, NULL);


    if ((expansion || smart) && (osTrapSwap(_NewHandle, NULL) != (smart ? &xaNewHandleSmart : &xaNewHandle)))

        // Replace the traps
        if (osTrapSwap(_NewHandle, smart ? &xaNewHandleSmart : &xaNewHandle) != gatTraps[ktrNewHandle])

//          Debugger();
//          osTrapSwap(_NewHandle, original);
//          SysBeep(9);


        if (expansion)

            if (osTrapSwap(_NewPtr, &xaNewPtr) != gatTraps[ktrNewPtr])

//              Debugger();
//              osTrapSwap(_NewPtr, original);
//              SysBeep(9);


            if (osTrapSwap(_DisposePtr, &xaDisposePtr) != gatTraps[ktrDisposePtr])

//              Debugger();
//              osTrapSwap(_DisposePtr, original);
//              SysBeep(9);



        if (smart)

            if (osTrapSwap(_OpenPicture, &xaOpenPicture) != gatTraps[ktrOpenPicture])

//              Debugger();
//              osTrapSwap(_DisposePtr, original);
//              SysBeep(9);



    SysProc trap = gatTraps[ktrMaxApplZone];

    LeaveCodeRsrc();

    return   trap;



// GLUE Code for tpNewHandle
#if         ISAPPC

static RoutineDescriptor xaMaxApplZone = BUILD_ROUTINE_DESCRIPTOR(bpiMaxApplZone, xaMaxApplZonePPC);

static pascal void xaMaxApplZonePPC(void)

    CallOSTrapUniversalProc(tpMaxApplZoneX(), bpiMaxApplZone);
```

```
#else    //   ISAPPC

static pascal asm void xaMaxApplZone(void)

     movem.l       d0-d2/a1, -(sp)
     jsr           tpMaxApplZoneX
     movem.l       (sp)+, d0-d2/a1
     jmp           (a0)


#endif   //   ISAPPC




yError tpInit(initBlk init)

 /* ____________________________________________________________
     tp        Init

     In        InitBlk from main
     Out

     Effect    Fill initBlk with our shit!

     Const
     Errors
     Flags
     Global
     Rsrc

     Version 001
     Notes
     History
     001 GOD 10.04.96 Use MaxApplZone
     000
 ____________________________________________________________ */
     trapTable                 traps;


     // Our Trap Table
     init->traps = traps = (trapTable)NewPtrSysClear(sizeof(TrapTable) + (sizeof(TrapTableItem) * (ktrCnt - 1)));

     if (!traps)
          return   memFullErr;

     traps->originalTraps = gatTraps;

     // The traps we patch
     int               cnt = 1;
     trapTableItem     t     = traps->item;

     t->trapWord = _Launch;
     t->agt      = &xaLaunch;

     ++ cnt;
     ++t;
     t->trapWord = _Get1Resource;
     t->agt      = &xaGet1Resource;
```

```
	++ cnt;
	++t;
	t->trapWord = _MaxApplZone;
	t->agt		= &xaMaxApplZone;

	// Just for inquiry
	++ cnt;
	++ t;
	t->trapWord = _OpenPicture;

	++ cnt;
	++ t;
	t->trapWord = _NewHandle;

	++ cnt;
	++ t;
	t->trapWord = _NewPtr;

	++ cnt;
	++ t;
	t->trapWord = _DisposePtr;

	// Our gestalts
	if (init->gestVals)
		init->gestVals->item[0].value = (long)&grp;

	// Set the number of traps we want
	traps->cnt = cnt;

	return	noErr;



#include	"mmBlkClr.c"
#include	"osTrapSwap.c"
```