```
//VEC OS
//#include <PalmOS.h>
//#include "VEC_OS_res.h"
#include "serial.h"
#include "supp.h"

#include "UCodes.h"
#include "B_ind.h"
#include "C_ind.h"
#include "P_ind.h"
/******
* Constants
******/
#define appFileCreator              'VECOS'
#define appVersionNum              0x01
#define appPrefID                  0x00
#define appPrefVersionNum          0x01
#define minVersion  sysMakeROMVersion(2,0,0,sysROMStageRelease,0)

#define RECEIVE_BLK_SIZE            10   //set this to 64
#define BAUD_RATE                   9600


/******
* Globals
******/
Boolean gSerialPortOpen;
UInt16  gSerialPortId;
Err     gSerialPortOpenErr;
Err     gSerialPortCloseErr;
Err     gSerialPortSendErr;
Err     gSerialPortReceiveErr;
int     gNumBytesSent;
UInt8   gBytesToReceive = RECEIVE_BLK_SIZE;
UInt32  gDataBuffer;
Char*   gReceiveBuffer;
Char    gReceiveData[RECEIVE_BLK_SIZE];
UInt8   gVolume;    //control bit to send out
UInt8   gSpeed;      //0=Vehicle Speed
UInt8   gNumDTCs;    //1=#DTCs
UInt8   gSensLR;     //2=Sensor1-Left Rear
UInt8   gSensRR;     //3=Sensor2-Right Rear
UInt8   gSensLS;     //4=Sensor3-Left Side
UInt8   gSensRS;     //5=Sensor4-Right Side
UInt8   gSensF;      //6=Sensor5-Front
UInt8   gELMerr;     //7=ELM error
UInt8   gWBerr;      //8=Winbond Error
UInt8   gSensErr;    //9=Sensor Error
UInt8   gDTC[36];    //10-45=DTC 1.1-18.2  :two bytes each
Char    gVIN[17];    //46-62 VIN#
UInt8   gInstance;
Boolean gDispLR;
```

```
Boolean gDispRR;
Boolean gDispLS;
Boolean gDispRS;
Boolean gDispF;
Boolean gFirst;

/******
* Prototypes
******/
Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags);
void * GetObjectPtr(UInt16 formID);
void MainFormInit(FormPtr /*frmP*/);
Boolean MenuCommand(UInt16 command);
Boolean MainFormHandleEvent(EventPtr eventP);
Boolean AppHandleEvent(EventPtr eventP);
void AppEventLoop(void);
Err AppStart(void);
void AppStop(void);static UInt32 StarterPalmMain(UInt16 cmd, MemPtr, UInt16 launchFlags);
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags);
void SerialReceive();
void Init();
void HandleDTCs();

/******
* Structs
******/
typedef struct
    {
    UInt8 prefType;
    } StarterPreferenceType;

typedef struct
    {
    UInt8 appInfoType;
    } StarterAppInfoType;

typedef StarterAppInfoType* StarterAppInfoPtr;

/******
* Functions
******/

/*********
 * handleDTCs - if there's a DTC reported, this allows it to be displayed
 *
 * returns:nothing
 *********/
void HandleDTCs()
{
    int test,tc;
    char* warning;
    char Error[] = "Code not found";
```

```
    for(int i=0;i<gNumDTCs;i=i+2){
        tc=gDTC[i]; //not sure if this will work or not

        test = tc & 0xF000;           // isolate the upper four bits to determine
                                      // which code set to use (P, C, B, or U)
        tc &= 0x0FFF;                 // isolate bits 0-5 to determine which code to use
        switch(test)
        {
            case 0x0000:
            case 0x1000:
            case 0x2000:
            case 0x3000:    if(tc > 0x2138) warning = Error;
                        else if(tc == 0x1501)
                        {
                            if(gVIN[BOD] == 'V') warning = &PCodes[30638];
                            else warning = &PCodes[30661];
                            break;
                        }
                        else if(tc == 0x1971)
                        {
                            if(gVIN[BOD] == 'J') warning = &PCodes[36363];
                            else warning = &PCodes[33630];
                        }
                        else if(tc == 0x1972)
                        {
                            if((gVIN[BOD] == 'V') || (gVIN[BOD] == 'Z')) warning = &PCodes
[36441];
                            else warning = &PCodes[33391];
                        }
                        else if(tc == 0x1890)
                        {
                            if((gVIN[YEAR] == 'V') || (gVIN[YEAR] == 'W')) warning = &
PCodes[38503];
                            else warning = &PCodes[38526];
                        }
                        else warning = P_ind[tc];    // P-code
                        break;

            case 0x4000:
            case 0x5000:
            case 0x6000:
            case 0x7000:    if(tc > 0x1788) warning = Error;
                        else if(tc == 0x0245)
                        {
                            if(gVIN[DIV] == 'C') warning = &CCodes[1301];
                            else warning = &CCodes[1328];
                        }
                        else warning = C_ind[tc];    // C-code
                        break;

            case 0x8000:
            case 0x9000:
```

```
            case 0xA000:
            case 0xB000:    if(tc == 0x1760)            // B-code
                        {
                            if(gVIN[BOD] == 'H') warning = &BCodes[8079];
                            else warning = &BCodes[5814];
                        }
                        else if(tc == 0x3642) warning = &BCodes[16221];
                        else if(tc == 0x3793) warning = &BCodes[16258];
                        else if(tc == 0x3833 || tc == 0x3832) warning = &BCodes[16290];
                        else if(tc == 0x3935) warning = &BCodes[16321];
                        else if(tc > 3527) warning = Error;
                        else warning = B_ind[tc];
                        break;
            case 0xC000:
            case 0xD000:
            case 0xE000:
            case 0xF000:                            // U-code
                        if(tc < 0x1000) warning = Error;
                        else
                        {
                            switch(tc)
                            {
                                case 0x1255:    // same as U1000
                                case 0x1000:    warning =   &UCodes[0]; break;
                                case 0x1001:    warning =   &UCodes[34];    break;
                                case 0x1016:    if(gVIN[BOD] == '5' || gVIN[BOD] == '6')
                                                    warning = &UCodes[84];
                                                else warning = &UCodes[123];
                                                break;
                                case 0x1026:    warning =   &UCodes[155];   break;
                                case 0x1040:    warning =   &UCodes[189];   break;
                                case 0x1041:    warning =   &UCodes[229];   break;
                                case 0x1042:    warning =   &UCodes[256];   break;
                                case 0x1056:    warning =   &UCodes[311];   break;
                                case 0x1064:    warning =   &UCodes[343];   break;
                                case 0x1065:    warning =   &UCodes[375];   break;
                                case 0x1066:    warning =   &UCodes[407];   break;
                                case 0x1088:    warning =   &UCodes[439];   break;
                                case 0x1096:    warning =   &UCodes[471];   break;
                                case 0x1128:    warning =   &UCodes[503];   break;
                                case 0x1129:    warning =   &UCodes[535];   break;
                                case 0x1144:    warning =   &UCodes[567];   break;
                                case 0x1145:    warning =   &UCodes[599];   break;
                                case 0x1146:    warning =   &UCodes[626];   break;
                                case 0x1152:    if(gVIN[BOD] == '1' || gVIN[BOD] == 'E'
                                                    || gVIN[BOD] == 'H' || gVIN[BOD] == 'J'
                                                    || gVIN[BOD] == 'K')    warning = &
UCodes[692];
                                                else warning = &UCodes[654];
                                                break;
                                case 0x1153:    warning =   &UCodes[717];   break;
                                case 0x1160:    warning =   &UCodes[749];   break;
```

```
                                    case 0x1161:    warning =   &UCodes[781];   break;
                                    case 0x1162:    warning =   &UCodes[810];   break;
                                    case 0x1163:    warning =   &UCodes[846];   break;
                                    case 0x1164:    warning =   &UCodes[880];   break;
                                    case 0x1166:    warning =   &UCodes[918];   break;
                                    case 0x1168:    warning =   &UCodes[950];   break;
                                    case 0x1169:    warning =   &UCodes[977];   break;
                                    case 0x1170:    warning =   &UCodes[1009];  break;
                                    case 0x1176:    warning =   &UCodes[1058];  break;
                                    case 0x1192:    warning =   &UCodes[1085];  break;
                                    case 0x1193:    warning =   &UCodes[1121];  break;
//                                  case 0x1255:    warning =   &UCodes[1155];  break;
                                    case 0x1300:    warning =   &UCodes[1184];  break;
                                    case 0x1301:    warning =   &UCodes[1208];  break;
                                    case 0x1304:    warning =   &UCodes[1233];  break;
                                    case 0x1305:    warning =   &UCodes[1261];  break;
                                    case 0x1500:    warning =   &UCodes[1291];  break;
                                    case 0x1715:    // Same as U1713
                                    case 0x1713:    warning =   &UCodes[1331];  break;
                                    case 0x1716:    // Same as U1714
                                    case 0x1714:    warning =   &UCodes[1390];  break;
                                    case 0x1800:    warning =   &UCodes[1451];  break;
                                    case 0x2000:    warning =   &UCodes[1499];  break;
                                    case 0x2001:    warning =   &UCodes[1532];  break;
                                    case 0x2002:    warning =   &UCodes[1565];  break;
                                    case 0x2003:    warning =   &UCodes[1582];  break;
                                    case 0x2004:    warning =   &UCodes[1600];  break;
                                    case 0x2005:    warning =   &UCodes[1618];  break;
                                    case 0x2006:    warning =   &UCodes[1637];  break;
                                    case 0x2007:    warning =   &UCodes[1656];  break;
                                    case 0x2050:    warning =   &UCodes[1676];  break;
                                    case 0x2100:    warning =   &UCodes[1720];  break;
                                    case 0x2102:    warning =   &UCodes[1755];  break;
                                    case 0x2103:    warning =   &UCodes[1795];  break;
                                    case 0x2104:    warning =   &UCodes[1836];  break;
                                    case 0x2105:    warning =   &UCodes[1866];  break;
                                    case 0x2106:    warning =   &UCodes[1903];  break;
                                    case 0x2107:    warning =   &UCodes[1939];  break;
                                    case 0x2108:    warning =   &UCodes[1984];  break;
                                    case 0x2150:    warning =   &UCodes[2018];  break;
                                    case 0x2151:    warning =   &UCodes[2061];  break;
                                    case 0x2153:    warning =   &UCodes[2102];  break;
                                    default:        warning =   &UCodes[34];    break;
                                }  // end switch
                            }  // end else
                        }//end switch


                    FrmCustomAlert(OBDAlert,warning,"","");


                }//end for
```

```
}
/*********
 * Init - initializes everything on main form
 *
 * returns:nothing
 *********/
void Init()
{
    FormPtr frmP;
    UInt8   ObjectIndex;
    frmP=FrmGetActiveForm();//get active form
    ObjectIndex=FrmGetObjectIndex(frmP,MainSerialConnectionBitMap);//get index of serial connection
bitmap
    FrmHideObject(frmP,ObjectIndex);
    gDispLR=true;
    gDispRR=true;
    gDispLS=true;
    gDispRS=true;
    gDispF=true;
    gInstance=0;
    gFirst=true;
    gVolume=0+48;
}
/*********
 * SerialReceive - receives data through serial port
 *
 * returns:nothing
 *********/
void SerialReceive()
{
    FormPtr frmP;
    UInt8   ObjectIndex;

    frmP=FrmGetActiveForm();//get active form
    ObjectIndex=FrmGetObjectIndex(frmP,MainSerialConnectionBitMap);//get index of serial connection
bitmap
    gSerialPortOpenErr = SrmOpen(serPortCradlePort,BAUD_RATE,&gSerialPortId);//open the serial port
    FrmShowObject(frmP,ObjectIndex);

    if (gSerialPortOpenErr == errNone)   //if there's no error opening port
    {
        gSerialPortOpen=true;            //set PortOpen = true
        //send volume control as synch byte
        gNumBytesSent = SrmSend(gSerialPortId, &gVolume, 1, &gSerialPortSendErr);
        if(gNumBytesSent != 1) FrmCustomAlert(SerialPortAlert,"Number of bytes sent does
not match intended length.","","");
        if(gSerialPortSendErr != errNone)
            SerialPortSendError(gSerialPortSendErr);
        else FrmAlert(SuccessAlert);

        UInt32 bytesReceived;
        gSerialPortReceiveErr = SrmReceiveWait(gSerialPortId,RECEIVE_BLK_SIZE,
```

```
SysTicksPerSecond()*6);
        if (gSerialPortReceiveErr != errNone)
            SerialPortReceiveError(gSerialPortReceiveErr,&gSerialPortId);
        else{
            gReceiveBuffer=gReceiveData;
            bytesReceived = SrmReceive(gSerialPortId,gReceiveBuffer,RECEIVE_BLK_SIZE,
SysTicksPerSecond()*6,&gSerialPortReceiveErr);
            if (gSerialPortReceiveErr != errNone){
                FrmCustomAlert(SerialPortAlert,"Problem with SrmReceive","","");
                SerialPortReceiveError(gSerialPortReceiveErr,&gSerialPortId);}
            else{
                FrmCustomAlert(SerialPortAlert,"The data has been received successfully.",
"","");
                gSpeed=gReceiveData[0];      //Vehicle speed
                displaySpeed(gSpeed);//display speed
                gNumDTCs=gReceiveData[1];    //#DTCs
                displayDTCs(gNumDTCs);//display # of DTCs

                gSensLR=gReceiveData[2];     //Sensor1-Left Rear
                gSensRR=gReceiveData[3];     //Sensor2-Right Rear
                gSensLS=gReceiveData[4];     //Sensor3-Left Side
                gSensRS=gReceiveData[5];     //Sensor4-Right Side
                gSensF=gReceiveData[6];      //Sensor5-Front
                gELMerr=gReceiveData[7];     //ELM error
                if(gELMerr && gELMerr!='0')      //if ELM error, display alert
                    FrmCustomAlert(SystemErrorAlert,"There is a problem with the ELM
circuit.  Make sure everything is connected properly.","","");
                gWBerr=gReceiveData[8];      //WinBond error
                if(gWBerr && gWBerr!='0')        //if WinBond error, display alert
                    FrmCustomAlert(SystemErrorAlert,"There is a problem with the WinBond
circuit.","","");
                gSensErr=gReceiveData[9];    //Sensor error
                if(gSensErr && gSensErr!='0')    //if there's a prob with any of the sensors
                    FrmCustomAlert(SystemErrorAlert,"There is a problem with one of the
sensors.","","");
    //            for(int i=0;i<37;i++){
    //                gDTC[i]=gReceiveData[10+i];}  //DTC 1.1-18.2, two bytes each
    //            for(int i=0;i<19;i++){
    //                ggVIN[i]=gReceiveData[46+i];}      //vehicles gVIN number

                SrmReceiveFlush (gSerialPortId,0); //flush the receive FIFO
        }
    }

    if (gSerialPortOpen)
        gSerialPortCloseErr = SrmClose(gSerialPortId);
    if (gSerialPortCloseErr == errNone) {
        gSerialPortOpen=false;   //port open = false
        FrmAlert(Success2Alert);
        FrmHideObject(frmP, ObjectIndex);}
    else SerialPortCloseError(gSerialPortCloseErr);
    }
```

```
    else SerialPortOpenError(gSerialPortOpenErr,&gSerialPortId);
}
/*********
 * RomVersionCompatible - checks that ROM version meets minimum requirements
 *
 * input:  requiredVersion - minimum rom version required
 *      launchFlags     - indicate if the application UI is initialized.
 *
 * returns:error code or zero if rom is compatible
 *********/
Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags)
{
    UInt32 romVersion;

    FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
    if (romVersion < requiredVersion)
    {
        if((launchFlags & (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp)) ==
            (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp))
        {
            FrmAlert (RomIncompatibleAlert);
            if(romVersion < minVersion)
            {
                AppLaunchWithCommand(sysFileCDefaultApp, sysAppLaunchCmdNormalLaunch, NULL
);
            }
        }
        return sysErrRomIncompatible;
    }
    return errNone;
}

/*********
 * GetObjectPtr - returns a pointer to an object in the current form.
 *
 * input:  formId - id of the form to display
 *********/
void * GetObjectPtr(UInt16 formID)
{
    FormPtr frmP;

    frmP = FrmGetActiveForm();
    return FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, formID));
}

/*********
 * MainFormInit - initializes the MainForm form.
 *
 * input:  frm - pointer to the MainForm form.
 *********/
void MainFormInit(FormPtr /*frmP*/)
{
```

```
}

/*********
 * MenuCommand - performs the menu command specified.
 *
 * input:  command  - menu item id
 *********/
Boolean MenuCommand(UInt16 command)
{
    Boolean handled = false;
    FormPtr frmP;

    switch (command)
    {
        case MainOptionsAbout:
            MenuEraseStatus(0);
            frmP = FrmInitForm (AboutForm);
            FrmDoDialog (frmP);
            FrmDeleteForm (frmP);
            handled = true;
            break;

        case MainOptionsVECTeam:
            MenuEraseStatus(0);
            frmP = FrmInitForm (VECTeamForm);
            FrmDoDialog (frmP);
            FrmDeleteForm (frmP);
            handled = true;
            break;
    }
    return handled;
}

/**********
 * MainFormHandleEvent - event handler for the MainForm
 *
 * input:  eventP  - a pointer to an EventType structure
 *
 * returns:true if event has handle and shouldn't be passed to higher level handler.
 *********/
Boolean MainFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr frmP;
    UInt8   ObjectIndex;

    switch (eventP->eType)
    {
        case menuEvent:
            return MenuCommand(eventP->data.menu.itemID);

        case frmOpenEvent:
```

```
            frmP = FrmGetActiveForm();
            MainFormInit(frmP);
            FrmDrawForm(frmP);
            handled = true;
            break;
        case ctlSelectEvent:
            if(eventP->data.ctlSelect.controlID == MainBtnExitButton)    //Exit button
            {
                if (gSerialPortOpen) gSerialPortCloseErr = SrmClose(gSerialPortId);
                eventP->eType = appStopEvent;
            }
            else if (eventP->data.ctlSelect.controlID == MainBtnDTCsButton) //Log Button
            {
                SerialReceive();
                //sdasdf
                //HandleDTCs();
            }
            else if (eventP->data.ctlSelect.controlID ==
MainSldrVolumeFeedbackSliderControl)
            {
                //UInt8 result;
                //result= (UInt8)eventP->data.ctlSelect.value;
                //displaySpeed(result); //for testing
                gVolume = (UInt8)eventP->data.ctlSelect.value;
                gVolume = gVolume+48;  //ascii char
            }
            handled = true;
            break;
        case nilEvent:
            if(!gFirst) Init();
            //test

            frmP=FrmGetActiveForm();//get active form
            //end test
            switch (gInstance){
                case 0:
                    //driver rear sensor
                    if(gSensLR==1 || gSensLR==2){
                        ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverRearBitMap);
                        if(gDispLR){
                            FrmHideObject(frmP,ObjectIndex);
                            gDispLR=false;
                        } else if (!gDispLR) {
                            FrmShowObject(frmP,ObjectIndex);
                            gDispLR=true;
                    }}
                    //passenger rear sensor
                    if(gSensRR==1 || gSensRR==2){
                        ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerRearBitMap);
                        if(gDispRR){
                            FrmHideObject(frmP,ObjectIndex);
                            gDispRR=false;
```

```
                    } else if (!gDispRR) {
                        FrmShowObject(frmP,ObjectIndex);
                        gDispRR=true;
                }}
                //driver middle sensor
                if(gSensLS==1 || gSensLS==2){
                    ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverMiddleBitMap);
                    if(gDispLS){
                        FrmHideObject(frmP,ObjectIndex);
                        gDispLS=false;
                    } else if (!gDispLS) {
                        FrmShowObject(frmP,ObjectIndex);
                        gDispLS=true;
                }}
                //passenger middle sensor
                if(gSensRS==1 || gSensRS==2){
                    ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerMiddleBitMap);
                    if(gDispRS){
                        FrmHideObject(frmP,ObjectIndex);
                        gDispRS=false;
                    } else if (!gDispRS) {
                        FrmShowObject(frmP,ObjectIndex);
                        gDispRS=true;
                }}
                //front sensor
                if(gSensF==1 || gSensF==2){
                    ObjectIndex=FrmGetObjectIndex(frmP,MainCarFrontBitMap);
                    if(gDispF){
                        FrmHideObject(frmP,ObjectIndex);
                        gDispF=false;
                    } else if (!gDispF) {
                        FrmShowObject(frmP,ObjectIndex);
                        gDispF=true;
                }}

                gInstance=1;
                break;
            case 1:
                //driver rear sensor
                if(gSensLR==2){
                    ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverRearBitMap);
                    if(gDispLR){
                        FrmHideObject(frmP,ObjectIndex);
                        gDispLR=false;
                    } else if (!gDispLR) {
                        FrmShowObject(frmP,ObjectIndex);
                        gDispLR=true;
                }}
                //passenger rear sensor
                if(gSensRR==2){
                    ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerRearBitMap);
                    if(gDispRR){
```

```
                    FrmHideObject(frmP,ObjectIndex);
                    gDispRR=false;
                } else if (!gDispRR) {
                    FrmShowObject(frmP,ObjectIndex);
                    gDispRR=true;
            }}
            //driver middle sensor
            if(gSensLS==2){
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverMiddleBitMap);
                if(gDispLS){
                    FrmHideObject(frmP,ObjectIndex);
                    gDispLS=false;
                } else if (!gDispLS) {
                    FrmShowObject(frmP,ObjectIndex);
                    gDispLS=true;
            }}
            //passenger middle sensor
            if(gSensRS==2){
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerMiddleBitMap);
                if(gDispRS){
                    FrmHideObject(frmP,ObjectIndex);
                    gDispRS=false;
                } else if (!gDispRS) {
                    FrmShowObject(frmP,ObjectIndex);
                    gDispRS=true;
            }}
            //front sensor
            if(gSensF==2){
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarFrontBitMap);
                if(gDispF){
                    FrmHideObject(frmP,ObjectIndex);
                    gDispF=false;
                } else if (!gDispF) {
                    FrmShowObject(frmP,ObjectIndex);
                    gDispF=true;
            }}

            gInstance=2;
            break;
        case 2:
            //receive data here
            gInstance=0;
            break;
    }

    if(gSensLR==0 && !gDispLR){ //if there's no longer a warning, stop blinking
        ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverRearBitMap);
        FrmShowObject(frmP,ObjectIndex);
    }
    if(gSensRR==0 && !gDispRR){ //if there's no longer a warning, stop blinking
        ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerRearBitMap);
        FrmShowObject(frmP,ObjectIndex);
```

```
            }
            if(gSensLS==0 && !gDispLS){  //if there's no longer a warning, stop blinking
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarDriverMiddleBitMap);
                FrmShowObject(frmP,ObjectIndex);
            }
            if(gSensRS==0 && !gDispRS){  //if there's no longer a warning, stop blinking
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarPassengerMiddleBitMap);
                FrmShowObject(frmP,ObjectIndex);
            }if(gSensF==0 && !gDispF){   //if there's no longer a warning, stop blinking
                ObjectIndex=FrmGetObjectIndex(frmP,MainCarFrontBitMap);
                FrmShowObject(frmP,ObjectIndex);
            }

            handled=true;
            break;

        default:
            break;
    }
    return handled;
}

/**********
 * AppHandleEvent - loads form resources and sets event handler for the form loaded.
 *
 * input:  event  - a pointer to an EventType structure
 *
 * returns:true if event has handle and shouldn't be passed to higher level handler.
 **********/
Boolean AppHandleEvent(EventPtr eventP)
{
    UInt16 formId;
    FormPtr frmP;

    if(eventP->eType == frmLoadEvent)
    {
        //load form resource
        formId = eventP->data.frmLoad.formID;
        frmP = FrmInitForm(formId);
        FrmSetActiveForm(frmP);

        // Set the event handler for the form.  The handler of the currently
        // active form is called by FrmHandleEvent each time is receives an
        // event.
        switch (formId)
        {
            case MainForm:
                FrmSetEventHandler(frmP, MainFormHandleEvent);
                break;

            default:
                break;
```

```
        }
        return true;
    }
    return false;
}

/**********
 * AppEventLoop - event loop for the application.
 *
 * no params
 **********/
void AppEventLoop(void)
{
    UInt16 error;
    EventType event;

    do
    {
        EvtGetEvent(&event, SysTicksPerSecond()/3); //3 times per second
        if(! SysHandleEvent(&event))
            if(! MenuHandleEvent(0,&event,&error))
                if(! AppHandleEvent(&event))
                    FrmDispatchEvent(&event);
    }while(event.eType != appStopEvent);
 }


/**********
 * AppStart - Gets current application's preferences.
 *
 * returns:Err value 0 if nothing went wrong
 **********/
Err AppStart(void)
{
    StarterPreferenceType prefs;
    UInt16 prefsSize;

    // Read the saved preferences / saved-state information.
    prefsSize = sizeof(StarterPreferenceType);
    if(PrefGetAppPreferences(appFileCreator, appPrefID, &prefs, &prefsSize, true) !=
        noPreferenceFound)
    {
    }
    return errNone;
}

/**********
 * AppStop - saves current state of the application.
 *
 * no params
 **********/
void AppStop(void)
```

```
{
    StarterPreferenceType prefs;

    // Write the saved preferences / saved-state information.  This data
    // will be backed up during a HotSync.
    PrefSetAppPreferences (appFileCreator, appPrefID, appPrefVersionNum, &prefs, sizeof(
prefs), true);

    FrmCloseAllForms();
}


/**********
 * StarterPalmMain - called by PilotMain when program is loaded.
 *
 * input:      cmd - extra commands
 *             launchFlags - indicate if the application UI is initialized
 **********/
UInt32 StarterPalmMain(UInt16 cmd, MemPtr, UInt16 launchFlags)
{
    Err error;

    error = RomVersionCompatible (minVersion, launchFlags);
    if(error) return (error);

    switch (cmd)
    {
        case sysAppLaunchCmdNormalLaunch:
            error = AppStart();
            if(error) return (error);

            FrmGotoForm(MainForm);
            //SerialInit();
            AppEventLoop();
            AppStop();
            break;

        default:
            break;
    }

    return errNone;
}

/**********
 * PilotMain - function that executes when program is loaded
 *
 * input:      cmd - extra commands
 *             cmdPBP - ptr to location in memory with code
 *             launchFlags - indicate if the application UI is initialized
 **********/
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
```

```
{
    return StarterPalmMain(cmd, cmdPBP,launchFlags);
}
```