Application Note

AN2385/D
Rev. 0, 11/2002

GPRS Wireless Internet Connectivity Using DragonBall™ and Palm OS Based PDAs

By *Lalan J. Mishra*

Contents

This paper describes and provides source code for a reference application, called *GSM-GPRS*, which provides mobile Internet access on a personal digital assistant (PDA) based on a DragonBall™ processor and the Palm OS operating system, via a Motorola general packet radio service (GPRS) enabled phone, the Timeport™ 280. Also, the paper provides guidelines for porting this application to other DragonBall-based PDA platforms.

1 Introduction

As GPRS-enabled wireless connectivity becomes more widely available, provided by a growing family of wireless data- and multimedia-centric devices, consumers are likely to trade in standalone devices for newer "always connected" offerings at increasing rates. However, many of the features of future wireless devices can be supported today with existing products.

For example, most PDAs support dial-up Internet access. PDAs can also run full HTML Web browsers that are easily available at little or no cost. These browsers provide the ability to surf any Web site, unlike the wireless application protocol (WAP) browsers available on some mobile phones, which can access only WAP-enabled Web sites. All that is needed to provide wireless Internet access on a standalone PDA is to connect the PDA to a mobile phone with wireless data connectivity, and provide software to enable the two devices to communicate. To establish the wireless data link between the PDA and an Internet service provider (ISP), either conventional mobile phones or GPRS-enabled mobile phones can be used. However, there is a marked difference between using a conventional mobile phone to establish a dial-up connection with an ISP and using a GPRS-enabled phone that provides "always on" connectivity for this purpose.

This paper illustrates the use of a DragonBall-powered Palm OS PDA with Motorola's Timeport™ 280 GPRS-enabled phone. The choice of these components is for illustration only. The concept can be implemented on any PDA platform using any GPRS-enabled phone.

2 Dial-up vs. GPRS Connection

This section describes the differences between using a dial-up connection and a GPRS connection to connect to the Internet using a mobile phone.

Both dial-up and GPRS connections have hardware and software components associated with them. The Timeport™ 280 mobile phone has a built-in modem, and this modem is the major hardware component connecting the client (in this case the PDA) with the Internet. The information (data stream) as received by the modem cannot be used directly by the PDA. Therefore, software implementing the point-to-point protocol (PPP) suite and a world-wide Web browser are required for both types of connection. PPP is a standard part of the Palm OS network stack and is available on all Web-enabled Palm OS PDAs. The browser can be installed as a separate application on the PDA.

This reference application uses the Eudora Web browser. The Eudora Internet Suite 2.1 for the Palm Computing Platform can be downloaded at no cost from the following location:
http://www.eudora.com/internetsuite/download/

The network setup of dial-up and GPRS connections for mobile Internet access on Palm OS look the same. The difference lies in the protocol implementation and the inner working of the two setups, which remain hidden to the end user. The difference perceived by the end user lies in the steps performed to establish the connection.

In the case of a dial-up connection, the user might need to create a specific dial-up profile consisting of information like user name, password, ISP's dial-up number, and so on. The PDA's built-in networking software then handles dialing the ISP's phone number, supplying the user name and password, and performing the PPP negotiations when prompted by the user. Once the connection is established, the user is informed by a message on the screen, signaling that the Web browser can be started.

The steps to establish a GPRS-based connection are slightly different. Having a GPRS-enabled phone does not ensure GPRS-based Internet access. The phone service provider must support the GPRS service and the user must subscribe to this service. Assuming that these conditions are met, establishing a connection involves sending a number of commands known as "AT commands" to the modem of the GPRS-enabled mobile phone. Using the PDA as it is, the user must create a lengthy setup script which must be run to establish the connection. Creation of this script (although a one-time job) is time consuming.

The *GSM-GPRS* application described in this paper simplifies the connection process and runs the Web browser automatically for GPRS-based Internet access. With this application, mobile Internet access is just a click away. Also, this application provides a dial key interface to initiate and receive mobile voice calls. This key implementation can be modified to create more elaborate phone applications.

3 *GSM-GPRS* Application

The *GSM-GPRS* application's main objective is to establish an Internet connection using the mobile phone's internal modem and then launch the Eudora Web browser programatically. In addition, the application can initiate and receive GSM voice calls.

For all communication with the mobile phone, the application uses the PDA's RS-232 serial communication port. Software flow control is used, and therefore RTS and CTS signal lines are used in addition to Tx, Rx, and GND lines for interface with the mobile phone. The speed of serial communication between the PDA and the mobile phone is set at 57.6 kbps.

3.1 Graphical User Interface (GUI)

The user interacts with the GUI for two purposes:

- To launch the GPRS-based Internet access
- To place and receive voice calls

The application does not provide features such as phone book, record of incoming and outgoing calls, and so on, on the PDA. However, the application can be modified by a programmer for such purposes.

Figure 1 illustrates the GUI command buttons. When a button is clicked, the PDA interacts with the mobile phone by sending modem AT commands.

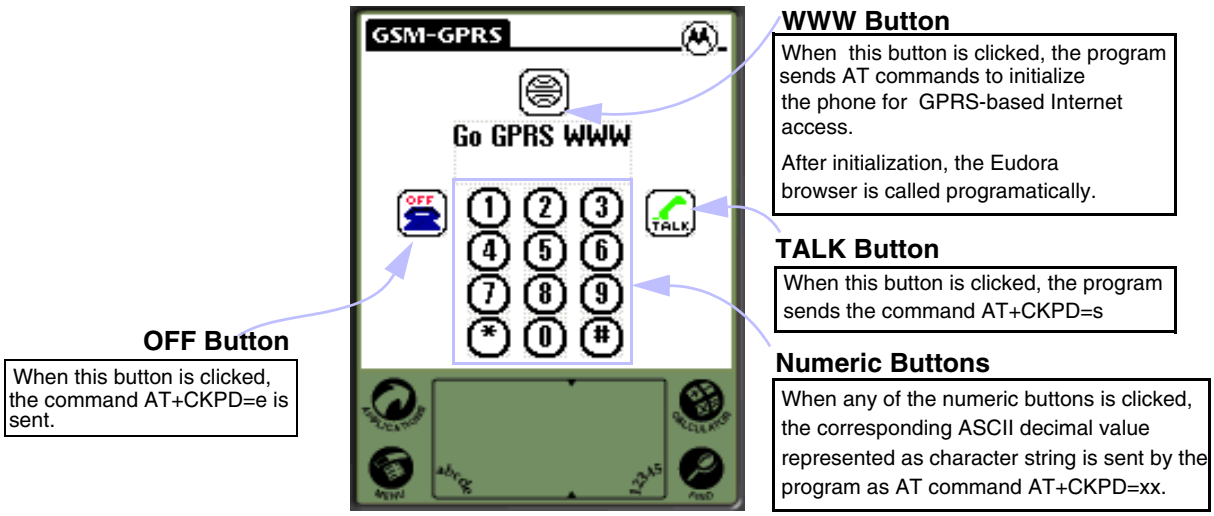

Figure 1. *GSM-GPRS* Application GUI

3.2 Modem AT Commands

Any data terminal equipment (DTE) device, when communicating with a modem, uses a set of commands to configure and control the modem. This set of commands is called AT commands. “AT” refers to the command prefix (ATtention sequence) that precedes each command to the modem. With the exception of “A/”, all commands must be preceded by “AT” and end with a carriage return (0x0D).

Detailed information on modem AT commands is widely available. This section explains the set of AT commands used in this application.

3.2.1 AT Commands Used in the *GSM-GPRS* Application

Three groups of AT commands are used in this application:

- Commands to emulate mobile phone’s key presses
- Commands to control and configure the modem
- Commands to do GPRS-related configuration

NOTE:

The *GSM-GPRS* application has been tested with the Motorola Timeport™ 280 phone. However, other Motorola GSM/GPRS-enabled phones use the same set of AT commands. Therefore, the application is expected to work without modification with those phones.

3.2.1.1 AT Command for Mobile Phone Key Emulation

When the PDA sends an phone key emulation AT command, the mobile phone device responds as if an actual key on its own keypad had been pressed. By implementing the AT commands for the entire set of keys on the mobile phone, a programmer can control the phone from the PDA through the RS-232 port.

The generic form of key emulation AT commands is: `AT+CKPD = xx <CR>`, where, *xx* represents the character string showing the decimal ASCII value of the key being emulated. <CR> represents the 1-byte ASCII value of carriage return, which is 0x0D.

For example, the decimal ASCII value of the character "1" is 49. Therefore, to emulate pressing the "1" key on the phone, the PDA must send: `AT+CKPD = 49 <CR>` to the mobile phone.

Table 1 shows the AT commands for key emulation used in this application.

Table 1. AT Commands for Mobile Phone Key Emulation

Button	AT Command String for Key Emulation
0	AT+CKPD = 48 <CR>
1	AT+CKPD = 49 <CR>
2	AT+CKPD = 50 <CR>
3	AT+CKPD = 51 <CR>
4	AT+CKPD = 52 <CR>
5	AT+CKPD = 53 <CR>
6	AT+CKPD = 54 <CR>
7	AT+CKPD = 55 <CR>
8	AT+CKPD = 56 <CR>
9	AT+CKPD = 57 <CR>
*	AT+CKPD = 42 <CR>
#	AT+CKPD = 35 <CR>
OFF	AT+CKPD = 69 <CR>
TALK	AT+CKPD = 83 <CR>

3.2.1.2 AT Commands for Modem Testing, Control, and Configuration

Although there is a large number of AT commands for modem control and configuration, this application needs only a few.

A total of 5 AT commands are used from this category. The actions that these commands perform include testing the presence of the modem, changing the communication baud rate, setting the data flow control mechanism, ignoring incoming calls, and dialing a given number.

Table 2 shows the AT commands belonging to this category.

Table 2. AT Commands for Modem Testing, Control, and Configuration

AT Commands	Meaning and Modem Response
AT <CR>	Detects the presence of the modem. The modem responds with **OK**.
ATS0 = 0 <CR>	Disables incoming calls.
ATE0	Turns modem echo off.

Table 2. AT Commands for Modem Testing, Control, and Configuration (Continued)

AT Commands	Meaning and Modem Response
AT+IPR = 57600 <CR>	Sets the modem baud rate to 57600 bps.
AT+IFC = 2, 2 <CR>	Sets the modem to RTS-CTS based hardware flow control.
ATD*99# <CR>	Dials the specified number. In this case the special number is *99# for the GPRS network.

3.2.1.3 AT Commands for GPRS-Related Configuration

In this application, a total of 4 AT commands are used for GPRS-related configuration. Table 3 shows the AT commands used for this purpose.

Table 3. AT Commands for GPRS-Related Configuration

AT Commands	Meaning and Command String
AT+CGATT?	Requests a GPRS attach (registration) on the GPRS network.
AT+CGDCONT=	Defines the packet data protocol (PDP) context. The command with the parameter used in this application is: `AT+CGDCONT = 1,"IP","internet2.voicestream.com", , 0,0 <CR>`
AT+CGQREQ =	Defines the quality of service profile. The command with the parameter used in this application is: `AT+CGQREQ = 1,0,0,3,0,0 <CR>`
AT+CGQMIN =	Defines the minimum acceptable quality of service profile. The command with the parameter used in this application is: `AT+CGQMIN = 1,0,0,3,0,0 <CR>`

NOTE:

The parameters used in the commands listed in Table 3 assume that the phone service provider is T-Mobile. Parameters for other service providers vary. For more information, refer to Section 6.1, “Porting to Another Service Provider” on page 18.

3.3 Application Architecture and Execution States

The application is based on a generic event-driven Palm OS application architecture. The GUI elements (buttons) are the sources of events. When the user clicks a button, a button-click event is generated. The application’s event handler then serves the event. As long as there are no events, the program waits for an event to occur.

All numeric, **OFF**, and **TALK** button events are served by sending a key emulation AT command to the phone device.

Pressing the **Go GPRS WWW** button initiates the configuration sequence of the modem for GPRS-based Internet access. When the configuration sequence is completed, the Eudora Web browser is launched programatically, without any user intervention. As soon as the Eudora browser is launched, the *GSM-GPRS* application closes automatically. Figure 2 shows *GSM-GPRS* application’s execution state diagram.

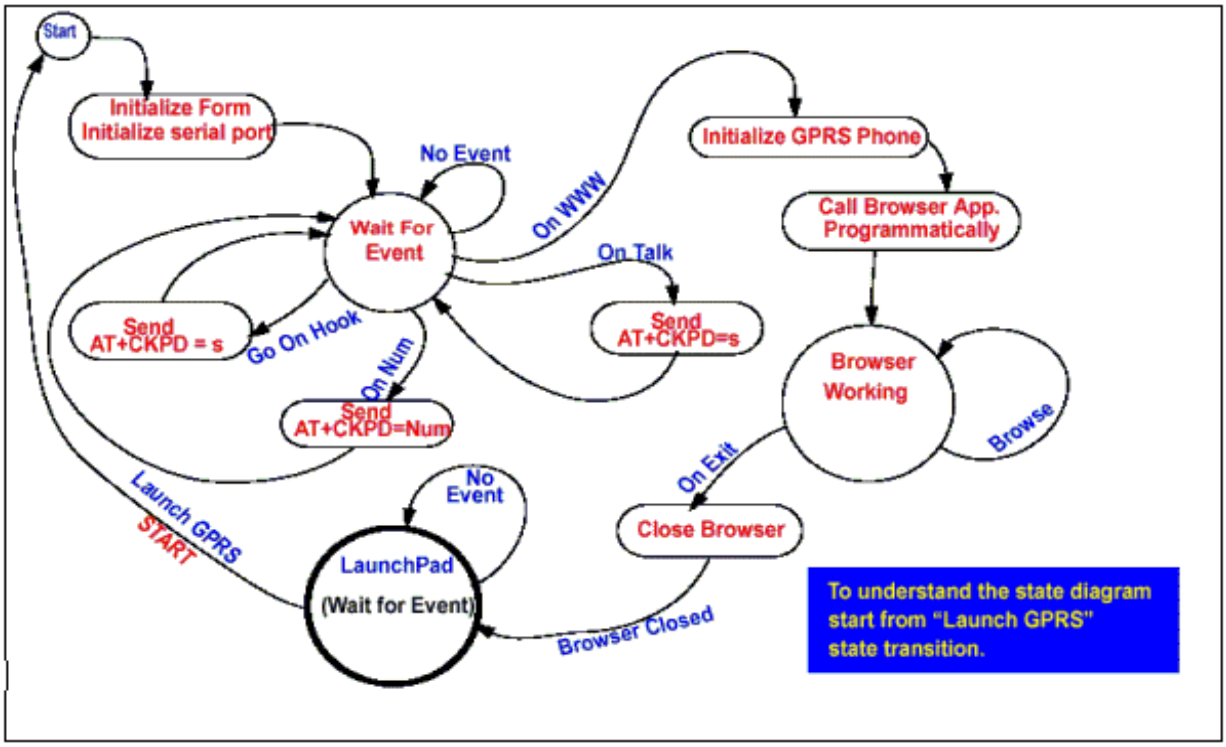

Figure 2. Execution State Diagram

The sequence of AT commands (with 250 ms delay between consecutive commands) that is sent to the mobile phone device on clicking **Go GPRS WWW** is shown in Figure 3. There is a 250 ms delay after every AT command.

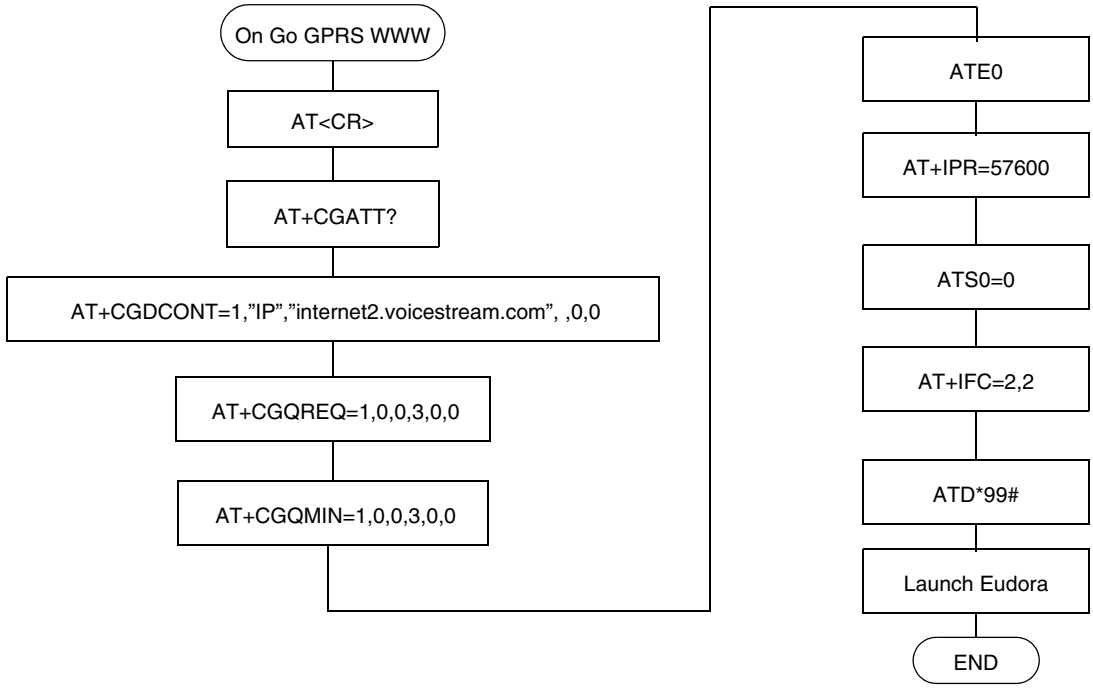

Figure 3. GPRS Initialization Flow Chart

4 Program Source Code

The *GSM-GPRS* application is a single form (window) application. Event handling is limited to responding to button-click events, and therefore is very simple. The code execution flow through various functions is shown in Figure 4.

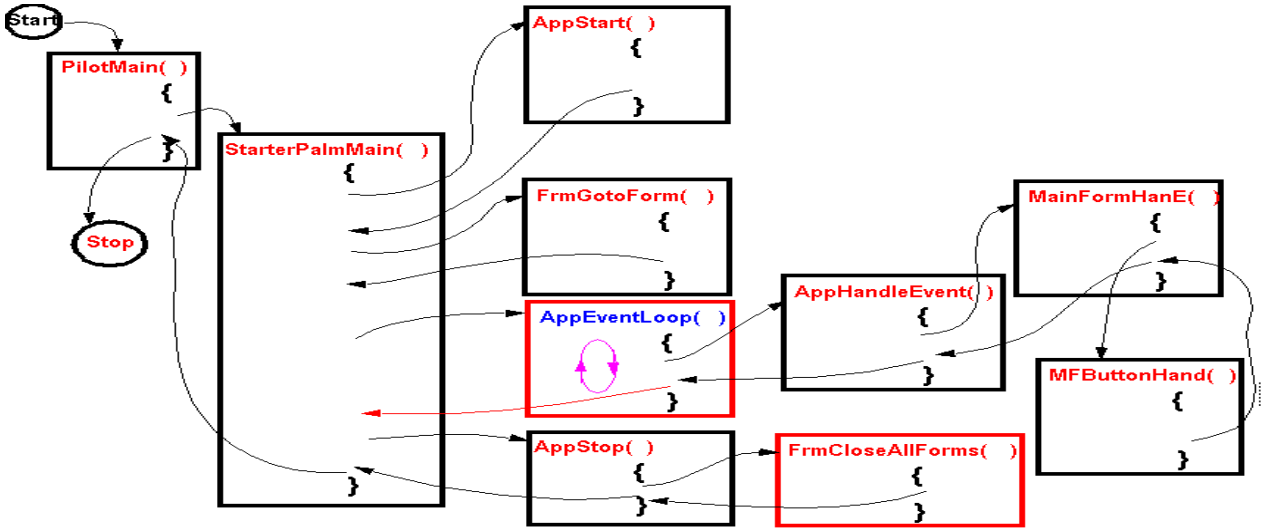

Figure 4. Code Execution Flow

The program was created using the standard Palm OS application template in Metrowerks CodeWarrior for Palm OS. The template was modified and renamed `Gprs-App.c`. This file is the main source file of the *GSM-GPRS* application. The header file `StarterRsc.h` contains the definitions for various objects used in the GUI. Other header files are standard Palm OS header files.

The source file can be viewed by opening the project file `GPRS.mcp` in the CodeWarrior environment. The contents of Gprs-App.c are shown in Code Listing 1.

Code Listing 1. Gprs-App.c

```
/*****************************************************************************
* Copyright Motorola Inc., Austin, Texas, U.S.A. All rights reserved.
* Description:    This file contains PilotMain() and other  functions to
*                 implement a basic GSM/GPRS dialer.
*
*Designed, coded and tested by:  Lalan J. Mishra (Lalan@motorola.com)
*Organization: Motorola Inc., Austin, Texas, (U.S.A.)
*
*****************************************************************************/

//---------------------------- Header File(s) Inclusion ------------------//
#include <PalmOS.h>
#include <SerialMgr.h>
#include "StarterRsc.h"
//------------------------------------------------------------------------//

//---------------------------- Internal Structures ----------------------//

typedef struct
      {
      UInt8 replaceme;
      } StarterPreferenceType;
```

```
typedef struct
      {
      UInt8 replaceme;
      } StarterAppInfoType;

typedef StarterAppInfoType* StarterAppInfoPtr;

//-------------------------------------------------------------------------//

//--------------------- Global Variables Follow ------------------------//
      Err err;
      UInt16 portId;
//-------------------------------------------------------------------------//

//---------------------------- Internal Constants ----------------------//
#define appFileCreator  'STRT'
#define appVersionNum              0x01
#define appPrefID                  0x00
#define appPrefVersionNum          0x01

                                        // Define the minimum OS version we
                                        // support (2.0 for now).
#define ourMinVersionsysMakeROMVersion(2,0,0,sysROMStageRelease,0)
#define extAppEudoraWeb 'QCwb'
//-------------------------------------------------------------------------//

//-------------------  Internal Function Prototypes ---------------------//
static void AppStop(void);
static Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags);
static void * GetObjectPtr(UInt16 objectID);
static Boolean MainFormDoCommand(UInt16 command);
static void MainFormInit(FormPtr);
static Err AppStart(void);
static UInt32 StarterPalmMain(UInt16 cmd, MemPtr, UInt16 launchFlags);
static void AppEventLoop(void);
static Boolean AppHandleEvent(EventPtr eventP);
static Boolean MainFormHandleEvent(EventPtr eventP);
static Boolean MainFormButtonHandler(FormPtr frmP, EventPtr  eventP);
//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    PilotMain                             //
// Objective:                                                              //
//              This is the MAIN ENTRY POINT for this application.         //
//-------------------------------------------------------------------------//

UInt32 PilotMain( UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
    return StarterPalmMain(cmd, cmdPBP, launchFlags);
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    StarterPalmMain                       //
// Objective:                                                              //
//              This is the ENTRY POINT for this application.              //
//-------------------------------------------------------------------------//

static UInt32 StarterPalmMain(UInt16 cmd, MemPtr /*cmdPBP*/, UInt16
launchFlags)
{
      Err error;

      error = RomVersionCompatible (ourMinVersion, launchFlags);
      if (error) return (error);
```

```
      switch (cmd)
            {
            case sysAppLaunchCmdNormalLaunch:
                  error = AppStart();
                  if (error)
                        return error;
                  FrmGotoForm(MainForm);
                  AppEventLoop();
                  AppStop();
                  break;

            default:
                  break;

            }

      return errNone;
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    AppStart                              //
// Objective:                                                              //
//               Get the current application's preferences.                //
// Called by:                                                              //
//           StarterPalmMain()                                             //
//-------------------------------------------------------------------------//

static Err AppStart(void)
{
    StarterPreferenceType prefs;
    UInt16 prefsSize;
    Boolean serPortOpened = false;

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

            }

      //FrmAlert(GPRSInitAlert);

        err = SrmOpen  (serPortCradlePort /* port */, 57600, /* baud */
                         &portId);

      return errNone;
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    AppStop                               //
// Objective:                                                              //
//               Save the current state of the application and exit.       //
// Called by:                                                              //
//           StarterPalmMain()                                             //
//-------------------------------------------------------------------------//

static 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);

      // Close all the open forms.
      SrmClose(portId);
      FrmCloseAllForms ();
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    AppEventLoop                          //
// Objective:                                                              //
//              This routine is the event loop for this application.       //
// Called by:                                                              //
//           StarterPalmMain()                                             //
//-------------------------------------------------------------------------//

static void AppEventLoop(void)
{
      UInt16 error;
      EventType event;

      do
            {
                  EvtGetEvent(&event, evtWaitForever);
                  if (! SysHandleEvent(&event))
                        if (! MenuHandleEvent(0, &event, &error))
                              if (! AppHandleEvent(&event))
                                    FrmDispatchEvent(&event);
            }
      while (event.eType != appStopEvent);
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                      Function:    AppHandleEvent                        //
// Objective:                                                              //
//           This routine loads form resources and set the event handler   //
//            for the form loaded.                                         //
// Called by:                                                              //
//           AppEventLoop()                                                //
//-------------------------------------------------------------------------//

static Boolean AppHandleEvent(EventPtr eventP)
{
      UInt16 formId;
      FormPtr frmP;

      if (eventP->eType == frmLoadEvent)
            {
            // Load the 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
            // it receives an event.
            switch (formId)
                  {
```

```
                  case MainForm:
                        FrmSetEventHandler(frmP, MainFormHandleEvent);
                        break;

                  default:
                        break;

                  }

            return true;
            }

      return false;
}

//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//                    Function : MainFormHandleEvent                       //
// Objective:                                                              //
//               This function is the event handler for the MainForm of    //
//               this application.                                          //
// Called by:                                                              //
//               AppHandleEvent()                                          //
//-------------------------------------------------------------------------//

static Boolean MainFormHandleEvent(EventPtr eventP)
{
   Boolean handled = false;
   FormPtr frmP;

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

            case ctlSelectEvent:
                  frmP = FrmGetActiveForm();
                  handled = MainFormButtonHandler(frmP, eventP);
                  break;

            case frmOpenEvent:
                  frmP = FrmGetActiveForm();
                  //MainFormInit( frmP);
                  FrmDrawForm ( frmP);
                  handled = true;
                  break;

            default:
                  break;

            }
      return handled;
}

//-------------------------------------------------------------------------//
```

```
//-----------------------------------------------------------------------//
//                Function : MainFormButtonHandler                       //
// Objective:                                                            //
//         This function processes the events generated by the buttons on//
//         the MainForm when they are pressed.                           //
// Called by:                                                            //
//         This function is called by MainFormHandleEvent() when the     //
//         Main Form Event Handler (MainFormHandleEvent()) finds that a  //
//         button press event has occured.                               //
//-----------------------------------------------------------------------//

static Boolean MainFormButtonHandler(FormPtr frmP, EventPtr  eventP)
{
      Boolean handled = false;

      UInt16cardNo;
      MemPtr Eud;
            LocalID dbID;
      char url[]="www.motorola.com";

      DmSearchStateType SearchState;

      char CarrRet[]="\r\n";
      char Dial[]="ATD*99#\r\n";

      UInt16 TxDelay=25, i;

      char Comma[]=",";
      char OnHook[]="AT+CKPD=e\r\n";// Go On Hook
      char Talk[]= "AT+CKPD=s\r\n";       // Talk

      char NumZero[]="AT+CKPD=48\r\n";// Num = 0
      char NumOne []="AT+CKPD=49\r\n";// Num = 1
      char NumTwo []="AT+CKPD=50\r\n";// Num = 2
      char NumThree[]="AT+CKPD=51\r\n";// Num = 3
      char NumFour[]="AT+CKPD=52\r\n";// Num = 4
      char NumFive[]="AT+CKPD=53\r\n";// Num = 5
      char NumSix[]="AT+CKPD=54\r\n";// Num = 6
      char NumSeven[]="AT+CKPD=55\r\n";// Num = 7
      char NumEight[]="AT+CKPD=56\r\n";// Num = 8
      char NumNine[]="AT+CKPD=57\r\n";// Num = 9

      char GprsInitMsgOne[]="AT\r\n";// AT<CR><NL>
      char GprsInitMsgTwo[]="AT+CGATT?\r\n";// AT+CGATT?<CR><NL>
      char GprsInitMsgThree[]="ATE0\r\n";// ATE0<CR><NL>

      char GprsInitMsgFour[]=
            "AT+CGDCONT=1,\"IP\",\"internet2.voicestream.com\",,0,0\r\n";

      char GprsInitMsgFive[]="AT+CGQREQ=1,0,0,3,0,0\r\n";//AT+CGQREQ
      char GprsInitMsgSix[]="AT+CGQMIN=1,0,0,3,0,0\r\n";//AT+CGQMIN

      char GprsInitMsgSeven[]="AT+IPR=57600\r\n";//AT+IPR=57600<CR><NL>
      char GprsInitMsgEight[]="ATZ\r\n";  //ATZ<CR><NL>
      char GprsInitMsgNine[]="ATE1V1\r\n";//ATE1V1<CR><NL>
      char GprsInitMsgTen[]="AT+IFC=2,2\r\n";//AT+IFC
      char GprsInitMsgEleven[]="ATS0=0\r\n";//ATS0
      char GprsInitMsgTwelve[]="AT+CGQREQ=1,0,0,3,0,0\r\n";//AT+CGQREQ

      // Finding which button was pressed.
      switch (eventP->data.ctlEnter.controlID)
            {
                  case MainPhoneOnHookGraphicButton:// Key "On_Hook"
                        SrmSend(portId, &OnHook, StrLen(OnHook), &err);
                        break;
```

```
case MainPhoneOffHookGraphicButton:// Key "Talk"
      SrmSend(portId, &Talk, StrLen(Talk), &err);
      break;

case MainKeyZeroButton:       // Key #0
      SrmSend(portId, &NumZero,StrLen(NumZero), &err);
      break;

case MainKeyOneButton:        // Key #1
      SrmSend(portId, &NumOne, StrLen(NumOne), &err);
      break;

case MainKeyTwoButton:        // Key #2
      SrmSend(portId, &NumTwo, StrLen(NumTwo), &err);
      break;

case MainKeyThreeButton:// Key #3
      SrmSend(portId, &NumThree, StrLen(NumThree), &err);
      break;

case MainKeyFourButton:       // Key #4
      SrmSend(portId, &NumFour, StrLen(NumFour), &err);
      break;

case MainKeyFiveButton:       // Key #5
      SrmSend(portId, &NumFive, StrLen(NumFive), &err);
      break;

case MainKeySixButton:        // Key #6
      SrmSend(portId, &NumSix, StrLen(NumSix), &err);
      break;

case MainKeySevenButton:// Key #7
      SrmSend(portId, &NumSeven, StrLen(NumSeven), &err);
      break;

case MainKeyEightButton:// Key #8
      SrmSend(portId, &NumEight, StrLen(NumEight), &err);
      break;

case MainKeyNineButton:       // Key #9
      SrmSend(portId, &NumNine, StrLen(NumNine), &err);
      break;

case MainKeyStarButton:       // Key "*"
      SndPlaySystemSound(7);
      break;

case MainKeyHashButton:       // Key "#"
      SndPlaySystemSound(7);
      break;

case MainInitGprsNetGraphicButton:// Key "GPRS_Init"

      // Tx >> "AT"
      for(i=0; i<=2; i++)
      {
      SrmSend(portId, &GprsInitMsgOne,
              StrLen(GprsInitMsgOne), &err);
      SysTaskDelay(TxDelay);
      SndPlaySystemSound(7);
      }

      // Tx >> "AT+CGATT?"
      SrmSend(portId, &GprsInitMsgTwo,
              StrLen(GprsInitMsgTwo), &err);
```

```
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "AT+CGDCONT=1,."
                        SrmSend(portId, &GprsInitMsgFour,
                                StrLen(GprsInitMsgFour), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "AT+CGQREQ=..."
                        SrmSend(portId, &GprsInitMsgFive,
                                StrLen(GprsInitMsgFive), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "AT+CGQMIN=..."
                        SrmSend(portId, &GprsInitMsgSix,
                                StrLen(GprsInitMsgSix), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "ATE0"
                        SrmSend(portId, &GprsInitMsgThree,
                                StrLen(GprsInitMsgThree), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "AT+IPR=57600"
                        SrmSend(portId, &GprsInitMsgSeven,
                                StrLen(GprsInitMsgSeven), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "ATS0=0"
                        SrmSend(portId, &GprsInitMsgEleven,
                                StrLen(GprsInitMsgEleven), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "AT+IFC=2,2"
                        SrmSend(portId, &GprsInitMsgTen,
                                StrLen(GprsInitMsgTen), &err);
                        SysTaskDelay(TxDelay);
                        SndPlaySystemSound(7);

                        // Tx >> "ATD*99#"
                        SrmSend(portId, &Dial, StrLen(Dial), &err);
                        SysTaskDelay(TxDelay*5);
                        SndPlaySystemSound(7);

                        //----- Programmatically launch Eudora Browser

                        DmGetNextDatabaseByTypeCreator (true, &SearchState,
                                                        sysFileTApplication,
                                                        extAppEudoraWeb, true,
                                                        &cardNo, &dbID);
                        Eud = MemPtrNew(StrLen(url)+1);
                        MemPtrSetOwner(Eud,0);
                        SysUIAppSwitch (cardNo, dbID,
                                        sysAppLaunchCmdNormalLaunch, Eud);
                        break;
                }
                handled = true;
                return handled;
}
//--------------------------------------------------------------------//
```

```
//--------------------------------------------------------------------------//
//                    Function : MainFormInit                               //
// Objective:                                                               //
//              This function initializes the MainForm form.                //
// Called by:                                                               //
//              MainFormHandleEvent()                                       //
//--------------------------------------------------------------------------//

static void MainFormInit(FormPtr)
{

}

//--------------------------------------------------------------------------//

//--------------------------------------------------------------------------//
//                    Function : MainFormDoCommand                          //
// Objective:                                                               //
//              This function performs the menu command specified.          //
// Called by:                                                               //
//              MainFormHandleEvent()                                       //
//--------------------------------------------------------------------------//

static Boolean MainFormDoCommand(UInt16 command)
{
      Boolean handled = false;
      FormPtr frmP;

      switch (command)
            {
            case MainOptionsAboutStarterApp:
                  MenuEraseStatus(0);     // Clear menu status from display
                  frmP = FrmInitForm (AboutForm);
                  FrmDoDialog (frmP);     // Display the About Box
                  FrmDeleteForm (frmP);
                  handled = true;
                  break;
            }

      return handled;
}
//--------------------------------------------------------------------------//

//--------------------------------------------------------------------------//
//                    Function : GetObjectPtr                               //
// Objective:                                                               //
//             This routine returns a pointer to an object in the current   //
//             form.                                                        //
//--------------------------------------------------------------------------//

static void * GetObjectPtr(UInt16 objectID)
{
      FormPtr frmP;

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

//--------------------------------------------------------------------------//
```

```
//--------------------------------------------------------------------------//
//                    Function : RomVersionCompatible                       //
// Objective:                                                               //
//               This routine checks that a ROM version meets the           //
//               minimum requirement.                                       //
// Called by:                                                               //
//               StarterPalmMain()                                          //
//--------------------------------------------------------------------------//

static Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags)
{
      UInt32 romVersion;

      // See if we're on in minimum required version of the ROM or later.
      FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
      if (romVersion < requiredVersion)
            {
            if ((launchFlags & (sysAppLaunchFlagNewGlobals |
                                sysAppLaunchFlagUIApp)) ==
                  (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp))
                  {
                  FrmAlert (RomIncompatibleAlert);

                  // Palm OS 1.0 continuously relaunches this app unless we
                  // switch to another safe one.
                  if (romVersion < ourMinVersion)
                        {
                        AppLaunchWithCommand(sysFileCDefaultApp,
                                             sysAppLaunchCmdNormalLaunch,
                                             NULL);
                        }
                  }

            return sysErrRomIncompatible;
            }

      return errNone;
}

//--------------------------------------------------------------------------//

//------------------------- End of File ------------------------------//
```

NOTE:

The CodeWarrior project file for the *GSM-GPRS* application can be downloaded as a `.zip` file `GPRS.zip` for further feature enhancement of the basic application. The ready to install *GSM-GPRS* application `.prc` file is located in the `Obj` subdirectory of the project installation directory.

5 Setting Up and Running the Application

To make the Motorola GPRS-enabled phone work in coordination with the PDA requires specific steps to be followed, described in the following sections:

- Loading the program
- Setting up the network connection
- Running the program

5.1 Loading the Program

The following program files should be loaded to the PDA:

- `Eudora.prc`
- `Eudora_Web.prc`
- `GPRS.prc`

These files can loaded using the HotSync operation or by any other means (such as IR beaming or copying from SD card). The GPRS application is identified by the Motorola logo.

5.2 Setting Up the Network Connection

Before attempting to connect to the Internet, you must complete the network settings on the PDA. This setup need only be performed once, using the following steps:

1. Select **Prefs** in the Applications list.
2. In the **Preferences** drop-down menu, click **Network**.
3. Set the **User Name** and **Password** as appropriate for your ISP account. For T-Mobile, the user name can be left blank, and the password can be any arbitrary string.
4. For **Connection**, select **Cradle/Cable**.
5. Click **Details**. The **Details** dialog box appears. Make the following settings:
   - Connection type: PPP
   - Idle Timeout: Never
   - Query DNS: select
   - IP address: select **Automatic**
6. Click **OK**.

5.3 Running the Program

Before you execute the **GSM-GPRS** program, you must connect the PDA and the Motorola phone.

Both the PDA and the Motorola phone come with serial (RS-232) cables for connection to a desktop computer. These cables are used to connect them to each other. However, both of these cables have 9-pin female connectors, so you need a null modem male-to-male connector to connect them to each other. A null modem connector can be obtained from most consumer electronics stores.

Once you connect the PDA and the phone, you can launch the *GSM-GPRS* program by clicking the program icon. As soon as the program is active, the PDA displays the screen shown in Figure 1 on page 3. From this screen, three operations are possible, as described in the following sections:

- Initiating a GSM call
- Receiving a GSM call
- Browsing the Internet over GPRS

5.3.1 Initiating a GSM Call

Click on the numeric buttons in the sequence of the number to be dialed. When a numeric button is clicked, the corresponding number appears on the mobile phone's screen. This operation also validates the connection between the PDA and the phone. After the entire number sequence has been sent to the phone, click **TALK**. This initiates the calling process and the call tones (busy or ring) can be heard on the phone. When the conversation is over, click **OFF** to end the call.

5.3.2 Receiving a GSM Call

Ringing indicates an incoming call. Click **TALK** to answer the call. When the conversation is over, click **OFF** to end the call.

5.3.3 Browsing the Internet over GPRS

To start GPRS-based Internet browsing, click **Go GPRS WWW**. The GPRS initialization process starts, and takes a short period of time to complete. When the initialization process is complete, the *GSM-GPRS* application terminates by launching the Eudora Web browser. Follow the on-screen instructions. When prompted, enter a URL and click **OK**. The Web page appears after a short period of time. Refer to the Eudora browser documentation for information on using the browser program.

To exit, use the **Exit Browser** option available in the Eudora browser application. If you exit the browser program in any other way, the phone remains in data mode, and must be turned off and on again in order to make and receive voice calls.

6 Guidelines for Porting

The *GSM-GPRS* program is designed to be simple, small, and easily ported. Porting options described in the following sections include:

- Porting to another service provider
- Porting to another operating system

6.1 Porting to Another Service Provider

The reference implementation of the *GSM-GPRS* program assumes that the phone service provider is T-Mobile. To port the application to another GPRS service provider, you must obtain two important parameters from the service provider. These are:

- Parameter string for the command AT+CGDCONT
- Parameter string for the command ATD

You can then change these parameter strings in the *GSM-GPRS* application source code. Compile and link the project to create a new `.prc` file.

6.2 Porting to Another Operating System

Porting this application to another DragonBall-based operating system, such as Symbian, Windows CE, or eLinux, requires rewriting the source code to fit the application framework of the new operating system, while maintaining the same program logic. The rewriting required is therefore substantial relative to the program size. However, because the program is very small, rewriting should not require substantial effort for a programmer who is already familiar with the target operating system.

In place of Metrowerks CodeWarrior for Palm OS, use a development tool designed for the target operating system.

7 Summary

GSM-GPRS is an application to automate GPRS-based Internet access on mobile PDA. The application uses a simple GUI and implements a minimal set of AT commands for interaction with a Motorola GPRS-enabled mobile phone. The application can be ported to other service provider and could also be ported to a platform running another OS by modifying the source code with no change in the program logic.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

AN2385/D