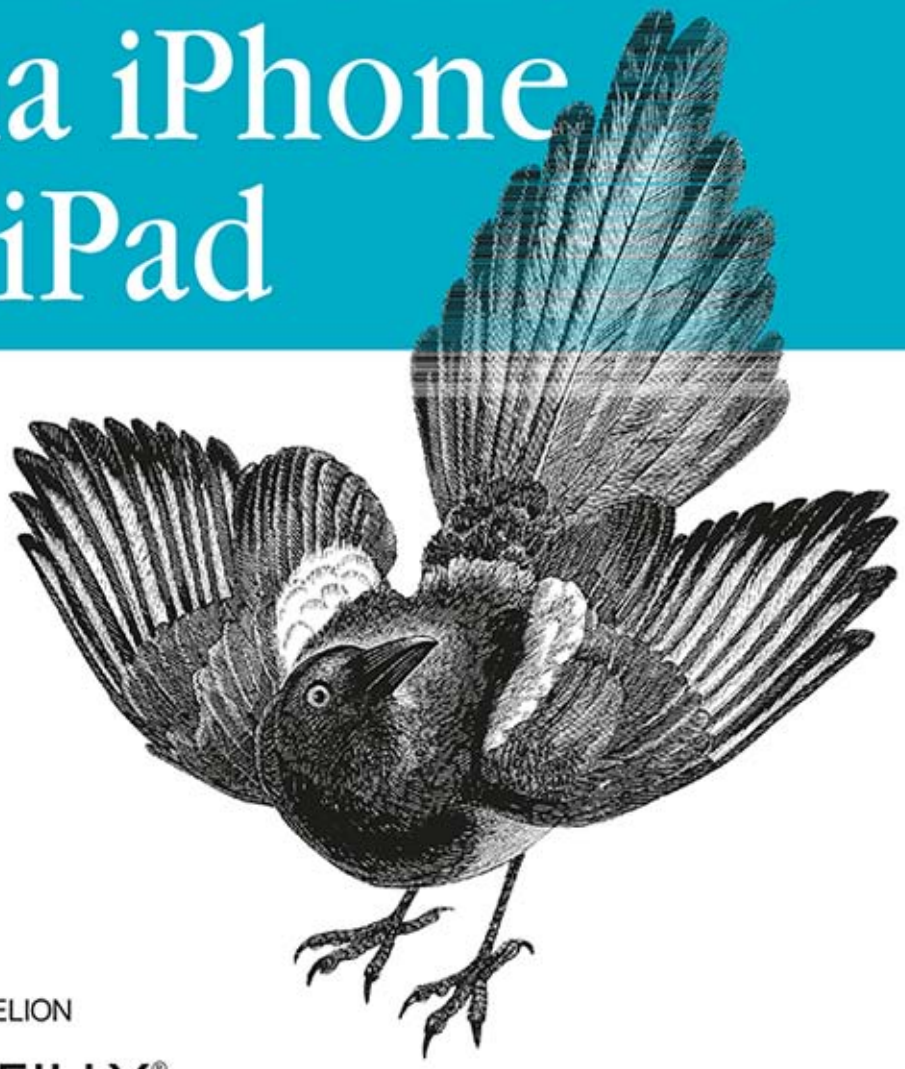
Niesamowite możliwości urządzeń z iOS!

Niekonwencjonalne gadżety z technologią Arduino i techBASIC

Projekty elektroniczne na iPhone i iPad



HELION

O'REILLY®

Mike Westerfield

Tytuł oryginału: Building iPhone and iPad Electronic Projects

Tłumaczenie: Robert Górczyński

ISBN: 978-83-246-8890-6

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/prelip
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Uzyskanie dostępu do innych wbudowanych czujników

W tym rozdziale

Przygotowania
Programy przedstawione w tym rozdziale są zmodyfikowanymi wersjami programu omówionego w rozdziale 1. Jeżeli natkniesz się na jakiekolwiek niejasności, wtedy zapoznaj się z rozdziałem 1.

Wyposażenie
Wymagane jest urządzenie iPhone, iPod touch bądź iPad działające pod kontrolą systemu iOS 5 lub nowszego.

Oprogramowanie
Wymagana jest kopia aplikacji techBASIC lub techBASIC Sampler.

Czego się nauczysz?
Z tego rozdziału dowiesz się, jak uzyskać dostęp do magnetometru i żyroskopu, które są wbudowane w większości urządzeń iOS. Wymienione czujniki można wykorzystać w praktycznie dowolnym celu, począwszy od wyznaczania kursu, aż po rzeczywistość rozszerzoną (ang. *Augmented Reality*).

Zanim zakończysz lekturę rozdziału, dowiesz się nieco więcej na temat techBASIC. Między innymi poznasz inny sposób uzyskania dostępu do czujników w znacznie krótszym czasie. Ponadto przekonasz się, że można używać systemu pomocy w aplikacji techBASIC do wyszukiwania informacji na temat samego języka i oferowanych przez niego poleceń.

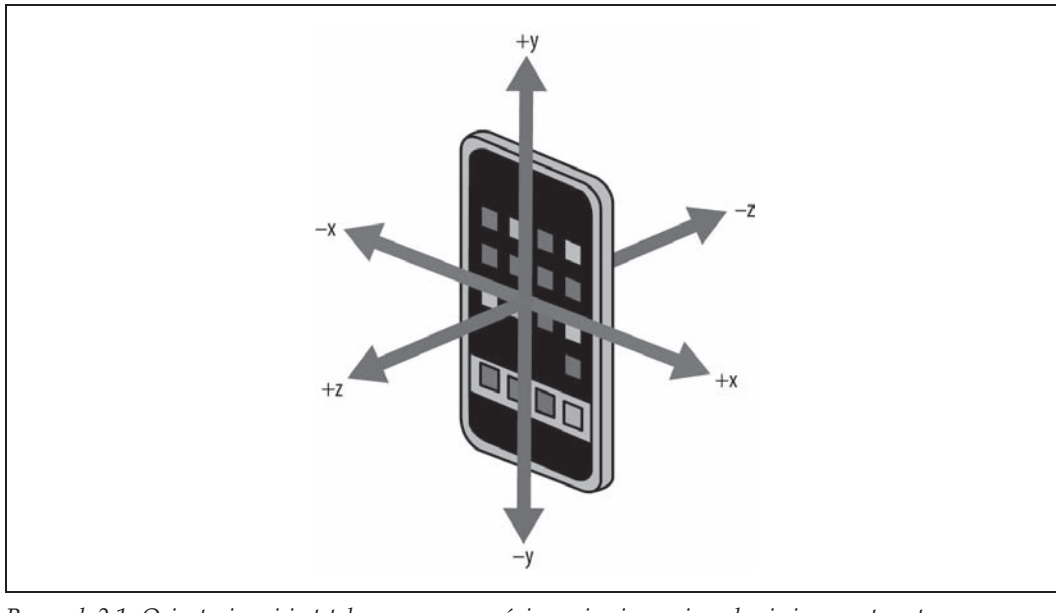
Na końcu rozdziału pokrótce zapoznasz się z dwoma innymi usługami. Wprawdzie to nie są typowe czujniki, ale dostęp do nich odbywa się w taki sam sposób jak do czujników. Pierwsza ze wspomnianych usług to GPS, natomiast druga to wyznaczanie kursu (ang. *heading*), które wykorzystuje magnetometr i kompas w celu znalezienia odpowiedniego kursu.

Utworzenie naszego pierwszego programu opartego na graficznym interfejsie użytkownika i przeznaczonego do wyświetlania danych czujnika wbudowanego w urządzenia iPhone i iPad wymagało nieco pracy. Nic dziwnego, poznajemy nowe środowisko programistyczne, a także

czujniki oferowane przez urządzenia iOS. Mając już podstawową wiedzę, warto ją rozszerzyć i wykorzystać nowe możliwości. Dzięki temu będziemy mogli ukończyć prace nad naszym tricorderem.

Żyroskop

Począwszy od modelu iPhone 4, wszystkie smartfony iPhone są wyposażone w żyroskop trójosiowy. Być może sądzisz, że żyroskop jest niepotrzebny, ponieważ przyśpieszeniomierz może dostarczyć wszelkich informacji o położeniu urządzenia (patrz rysunek 2.1), śledzić położenie i informować o wszelkich zmianach orientacji. Okazuje się jednak, że przyśpieszeniomierz nie sprawdza się jako zamiennik żyroskopu. Jednym z powodów są prawa fizyki. Grawitacja Ziemi to nie jedyne przyśpieszenie wychwytywane przez przyśpieszeniomierz, sam ruch także wiąże się z pewnym przyśpieszeniem. Inny powód jest czysto praktyczny. Przyśpieszeniomierz nie może wykrywać gwałtownych zmian w orientacji tak dobrze jak żyroskop, który został opracowany właśnie do tego celu.



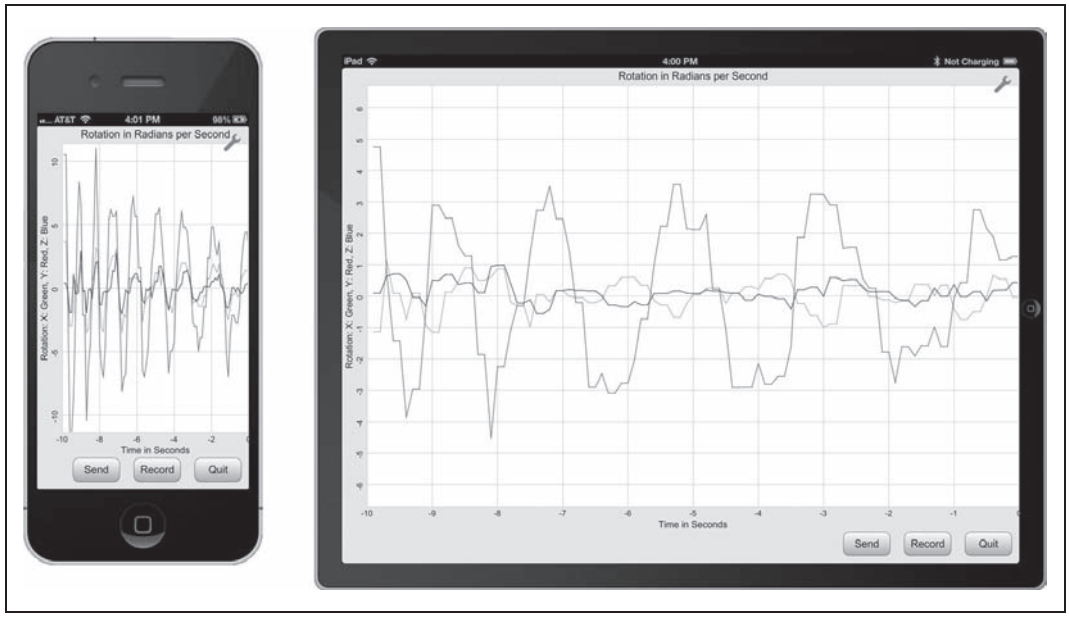
Rysunek 2.1. Orientacja osi jest taka sama w przyśpieszeniomierzu, żyroskopie i magnetometrze

Interfejs użytkownika aplikacji żyroskopu, którą utworzymy w tym rozdziale (patrz rysunek 2.2), jest bardzo podobny do aplikacji przyśpieszeniomierza omówionej w poprzednim rozdziale.

Kod źródłowy aplikacji również jest bardzo podobny do aplikacji omówionej w poprzednim rozdziale. Oczywiście między wspomnianymi aplikacjami występują pewne różnice, ale tworzonego tutaj programu nie będziemy omawiać wiersz po wierszu. Zamiast tego przyjrzymy się dzielącym je różnicom.

Pierwsza i najbardziej oczywista zmiana dotyczy czujnika, z którego będą pobierane wartości. W programie nie pobieramy wartości z czujnika przyśpieszeniomierza:

```
PRINT Sensors.accel
```



Rysunek 2.2. Uruchomiona aplikacja Gyroscope

ale z czujnika żyroskopu, korzystając z bardzo podobnego polecenia:

```
PRINT Sensors.gyro
```

W programie są jeszcze inne subtelne zmiany. Czy wiesz, że żyroskop jest dostępny tylko w niektórych modelach smartfona iPhone? Oznacza to, że bardzo ważne jest sprawdzenie dostępności czujnika, zanim będzie można przystąpić do odczytywania jego danych. Z tego powodu w dwóch miejscach programu wprowadzono odpowiednie zmiany. Pierwsza zmiana znajduje się już na początku programu, w kodzie konfiguracyjnym odpowiedzialnym za pobranie wartości początkowej czasu:

```
! Pobranie i ustawienie wartości początkowej czasu dla żyroskopu.
DIM t0 AS DOUBLE
IF Sensors.gyroAvailable THEN
  WHILE t0 = 0
    r = Sensors.gyro
    t0 = r(4)
  WEND
END IF
```

Polecenie `IF` sprawdza dostępność żyroskopu, zanim program spróbuje odczytać z niego dane. Kod zawiera także pętlę `WHILE`, aby zagwarantować pobranie niezerowej wartości czasu. Powód jest prosty: zanim nie osiągnie pełnej gotowości do pracy, żyroskop może zwrócić wartość zero.

Druga zmiana została wprowadzona na końcu podprocedury `setUpGUI`:

```
! Upewnienie się o dostępności żyroskopu. Jeżeli czujnik jest niedostępny,
! należy wyświetlić odpowiedni komunikat i zakończyć działanie programu.
IF NOT Sensors.gyroAvailable THEN
  msg$ = "This device does not have a gyroscope. "
  msg$ = msg$ & "The program will exit."
  button = Graphics.showAlert("No Gyro", msg$)
  STOP
END IF
```

Powyższy fragment kodu sprawdza, czy urządzenie jest wyposażone w żyroskop. W przypadku jego braku następuje wyświetlenie odpowiedniego komunikatu i zakończenie działania programu.

Inna drobna zmiana została wprowadzona na początku podprocedury `setUpGUI`. Polecenie ustawiające częstotliwość pobierania danych z czujnika musi być zmodyfikowane, aby zdefiniować częstotliwość dla żyroskopu, a nie przyśpieszeniomierza. Poniżej przedstawiono nowy fragment kodu:

```
SUB setUpGUI
! Pobieranie wartości z żyroskopu co 0,05 sekundy.
Sensors.setGyroRate(0.05)
```

Po uruchomieniu programu możesz zauważyć, że wykres jest obsługiwany w nieco odmienny sposób. W aplikacji odczytującej dane przyśpieszeniomierza po narysowaniu wykresu do końca jego rysowanie ponownie zaczyna się od początku. Z kolei aplikacja żyroskopu nieustannie uaktualnia wykres, przesuwając istniejące punkty w lewą stroną i umieszczając nowe po prawej stronie. Takie rozwiązanie wymaga wprowadzenia dwóch zmian w programie. Pierwsza to usunięcie zmiennej `index`, ponieważ nie jest dłużej potrzebna. Druga zmiana dotyczy podprocedury `nullEvent`, w której kod należy zmienić z:

```
ax(index, 2) = a(1)
ay(index, 2) = a(2)
az(index, 2) = a(3)
index = index + 1
IF index > 100 THEN index = 1
```

na następujący:

```
FOR i = 1 TO 99
   rx(i, 2) = rx(i + 1, 2)
   ry(i, 2) = ry(i + 1, 2)
   rz(i, 2) = rz(i + 1, 2)
NEXT
rx(100, 2) = r(1)
ry(100, 2) = r(2)
rz(100, 2) = r(3)
```

Możesz zadawać sobie pytanie, czy skopiowanie 297 wartości z jednego miejsca w tablicy do innego nie zabiera większej ilości czasu niż po prostu uaktualnienie jednej wartości i przejście dalej. Masz rację, to wymaga więcej czasu, ale dla nowoczesnego procesora tego rodzaju zmiana naprawdę nie powoduje widocznego pogorszenia wydajności działania programu. Operacja kopiowania wartości tablicy jest przeprowadzana bardzo szybko. Największy spadek wydajności jest związany z uaktualnianiem zawartości wyświetlanej na ekranie. To niewątpliwie należy wziąć pod uwagę, ale po uruchomieniu programu przekonasz się, że spadek wydajności jest niezauważalny.

Wiele opcji

W aplikacjach przyśpieszeniomierza i żyroskopu uaktualnienia wykresu są obsługiwane na odmienne sposoby. Który z nich uważasz za lepszy? Jeżeli preferujesz jeden z nich, wtedy bardzo łatwo możesz zmodyfikować program i otrzymać żądany sposób uaktualniania wykresu.

Ponieważ po prawej stronie wykresu program wyświetla wartość w aktualnej sekundzie, a następnie w lewą stronę umieszczane są dane z poprzednich dziesięciu sekund, zakres dla osi X został zmieniony z od 0 do 10 na od –10 do 0. Konieczne jest więc wprowadzenie

dwóch zmian w podprocedurze `setUpGUI`. Pierwsza zmiana dotyczy kodu odpowiedzialnego za początkowe przypisywanie wartości *x* w tablicach wykresu:

```
! Inicjalizacja tablic używanych przez wykres.
FOR t = 1 TO 100
  rx(t, 1) = t/10.0 - 10
  ry(t, 1) = t/10.0 - 10
  rz(t, 1) = t/10.0 - 10
NEXT
```

Druga zmiana dotyczy widocznego zakresu wykresu po jego inicjalizacji. Zamiast wyświetlać wartości od 0 do 10 dla osi X, program wyświetla wartości od –10 do 0.

```
! Ustawienie zakresu wykresu i domeny. To trzeba zrobić
! po dodaniu pierwszego obiektu PlotPoint, ponieważ ten krok
! również powoduje ustawienie zakresu i domeny.
p.setView(-10, -10, 0, 10, 0)
```

Jeżeli porównujesz kod źródłowy aplikacji Accelerometer i Gyroscope, to prawdopodobnie zauważyłeś, że zmiany są czysto kosmetyczne. Dotyczą nazw zmiennych, komentarzy i etykiet, co ma na celu odwoływanie się do żyroskopu, a nie przyśpieszeniomierza.

Poniżej przedstawiono pełny kod źródłowy aplikacji odczytującej dane z żyroskopu. Ten program o nazwie Gyroscope znajdziesz również w katalogu *O'Reilly Books* w aplikacjach techBASIC i techBASIC Sampler:

```
! Program wyświetla wykresy wartości pobranych z żyroskopu w ciągu
! ostatnich 10 sekund w odstępach co 0,1 sekundy. Program oferuje funkcję rejestracji
! pobieranych wartości i wysłania wyników za pomocą poczty elektronicznej.

! Utworzenie wykresów i tablic przechowujących punkty wykresów.
DIM p as Plot, px as PlotPoint, py as PlotPoint, pz as PlotPoint
DIM rx(100, 2), ry(100, 2), rz(100, 2)

! Utworzenie kontrolek.
DIM quit AS Button, record AS Button, send AS Button

! Utworzenie i inicjalizacja zmiennych globalnych używanych do śledzenia przebiegu działania programu.
fileName$ = "tempdata.txt"
recording = 0

! Pobranie i ustawienie wartości początkowej czasu dla żyroskopu.
DIM t0 AS DOUBLE
IF Sensors.gyroAvailable THEN
  WHILE t0 = 0
    r = Sensors.gyro
    t0 = r(4)
  WEND
END IF

! Utworzenie interfejsu użytkownika.
setUpGUI

! Utworzenie nowego przycisku i wypełnienie go gradientem.
!
! Parametry:
!   x - położenie poziome.
!   y - położenie pionowe.
!   title - tekst wyświetlany przez przycisk.
!
! Wartość zwrotna: nowy przycisk.
FUNCTION newButton (x, y, title AS STRING) AS Button
DIM b AS Button
```

```
b = Graphics.newButton(x, y)
b.setTitle(title)
b.setBackgroundColor(1, 1, 1)
b.setGradientColor(0.6, 0.6, 0.6)
newButton = b
END FUNCTION

! Ta podprocedura jest wywoływana, gdy nic innego się nie dzieje.
! Sprawdza, czy od ostatniego odczytania wartości żyroskopu
! upłynęło już co najmniej 0,1 sekundy. Jeśli tak, następuje wówczas
! pobranie i wyświetlenie nowych danych.
!
! Parametry:
!   time - godzina wystąpienia zdarzenia.
SUB nullEvent (time AS DOUBLE)
r = Sensors.gyro

IF recording AND (t0 <> r(4)) THEN
  PRINT #1, r(1); ","; r(2); ","; r(3); ","; r(4)
END IF

IF r(4) > t0 + 0.1 THEN
  WHILE r(4) > t0 + 0.1
    t0 = t0 + 0.1
    FOR i = 1 TO 99
      rx(i, 2) = rx(i + 1, 2)
      ry(i, 2) = ry(i + 1, 2)
      rz(i, 2) = rz(i + 1, 2)
    NEXT
    rx(100, 2) = r(1)
    ry(100, 2) = r(2)
    rz(100, 2) = r(3)
  WEND
  px.setPoints(rx)
  py.setPoints(ry)
  pz.setPoints(rz)
END IF
END SUB

! Ostatnio zarejestrowane dane będą wysłane za pomocą wiadomości e-mail.
SUB sendData
DIM e AS eMail
e = System.newEMail
IF e.canSendMail THEN
  e.setSubject("Gyroscope data")
  e.setMessage("Gyroscope data")
  e.addAttachment(fileName$, "text/plain")
  e.send
ELSE
  button = Graphics.showAlert("Can't Send",
     "Email cannot be sent from this device.")
END IF
END SUB

! Konfiguracja interfejsu użytkownika.
SUB setUpGUI

! Pobieranie wartości z żyroskopu co 0,05 sekundy.
Sensors.setGyroRate(0.05)

! Inicjalizacja tablic używanych przez wykres.
FOR t = 1 TO 100
  rx(t, 1) = t/10.0 - 10
```

```
  ry(t, 1) = t/10.0 - 10
  rz(t, 1) = t/10.0 - 10
NEXT

! Inicjalizacja wykresu i jego wyświetlenie.
p = Graphics.newPlot
p.setTitle("Rotation in Radians per Second")
p.setXAxisLabel("Time in Seconds")
p.setYAxisLabel("Rotation: X: Green, Y: Red, Z: Blue")
p.showGrid(1)
p.setGridColor(0.8, 0.8, 0.8)
p.setAllowedGestures($0042)

px = p.newPlot(rx)
px.setColor(0, 1, 0)
px.setPointColor(0, 1, 0)

py = p.newPlot(ry)
py.setColor(1, 0, 0)
py.setPointColor(1, 0, 0)

pz = p.newPlot(rz)
pz.setColor(0, 0, 1)
pz.setPointColor(0, 0, 1)

! Ustawienie zakresu wykresu i domeny. To trzeba zrobić
! po dodaniu pierwszego obiektu PlotPoint, ponieważ ten krok
! również powoduje ustawienie zakresu i domeny.
p.setView(-10, -10, 0, 10, 0)

! Wyświetlenie widoku przedstawiającego wykresy. Przekazanie wartości 1
! dla parametru powoduje przejście do trybu pełnego ekranu.
system.showGraphics(1)

! Zablokowanie ekranu w jego bieżącej orientacji.
orientation = 1 << (System.orientation - 1)
System.setAllowedOrientations(orientation)

! Określenie wymiarów wykresu.
p.setRect(0, 0, Graphics.width, Graphics.height - 47)

! Wyświetlenie tła.
Graphics.setPixelGraphics(0)
Graphics.setColor(0.886, 0.886, 0.886)
Graphics.fillRect(0, 0, Graphics.width, Graphics.height)

! Konfiguracja interfejsu użytkownika.
h = Graphics.height - 47
quit = newButton(Graphics.width - 82, h, "Quit")
record = newButton(Graphics.width - 174, h, "Record")
send = newButton(Graphics.width - 266, h, "Send")

! Jeżeli nie ma żadnych danych do wysłania, wtedy należy wyłączyć przycisk Send.
IF NOT EXISTS(fileName$) THEN
  send.setEnabled(0)
END IF

! Upewnienie się o dostępności żyroskopu. Jeżeli czujnik jest niedostępny,
! należy wyświetlić odpowiedni komunikat i zakończyć działanie programu.
IF NOT Sensors.gyroAvailable THEN
  msg$ = "This device does not have a gyroscope. "
  msg$ = msg$ & "The program will exit."
```

```
  button = Graphics.showAlert("No Gyro", msg$)
  STOP
END IF
END SUB

! Podprocedura wywoływana, gdy program powinien rozpocząć
! rejestrację danych. Ta podprocedura zmienia nazwę przycisku
! na Stop, otwiera plik danych wyjściowych, a następnie ustawia
! flagę wskazującą na konieczność rejestracji danych pobieranych
! z żyroskopu.
SUB startRecording
record.setTitle("Stop")
recording = 1
OPEN fileName$ FOR OUTPUT AS #1
END SUB

! Podprocedura wywoływana, gdy program powinien zakończyć
! rejestrację danych. Ta podprocedura zmienia nazwę przycisku
! na Record, zeruje wartość zmiennej recording, a następnie zamyka
! plik danych wyjściowych.
!
! Tę podprocedurę można bezpiecznie wywołać, nawet
! jeśli nie zostały zarejestrowane żadne dane.
SUB stopRecording
IF recording THEN
  record.setTitle("Record")
  CLOSE #1
  recording = 0
  send.setEnabled(1)
END IF
END SUB

! Obsługa naciśnięcia dowolnego przycisku.
!
! Parametry:
!   ctrl - przycisk, który został naciśnięty.
!   time - godzina, o której wystąpiło zdarzenie.
SUB touchUpInside (ctrl AS Button, time AS DOUBLE)
IF ctrl = quit THEN
  stopRecording
  STOP
ELSE IF ctrl = record THEN
  IF recording THEN
    stopRecording
  ELSE
    startRecording
  END IF
ELSE IF ctrl = send THEN
  stopRecording
  sendData
END IF
END SUB
```

Radiany czy stopnie?

Osoby zajmujące się fizyką, inżynierowie i matematycy bardzo często posługują się radianami i nie mają żadnych problemów w używaniu naturalnej jednostki obrotu, jaką w przypadku żyroskopu są radiany na sekundę. Jeśli jednak Twoją reakcją jest „radi-co?", to śpieszę wyjaśnić, że bardzo łatwo można zmodyfikować program, aby wartości obrotu były wyrażane w stopniach na sekundę zamiast w radianach na sekundę. Wartość odczytaną z czujnika

wystarczy pomnożyć przez 180/π, co spowoduje jej konwersję z radianów na stopnie. W języku techBASIC znajduje się nawet użyteczna funkcja o nazwie `DEG` przeznaczona do wspomnianej konwersji. Najlepszym rozwiązaniem jest przeprowadzenie konwersji tuż po odczytaniu wartości z czujnika. W podprocedurze `nullEvent` umieść więc pętlę `FOR`:

```
SUB nullEvent (time AS DOUBLE)
r = Sensors.gyro

FOR i = 1 TO 3
  r(i) = DEG(r(i))
NEXT
```

Oczywiście zmianie ulegnie także zakres wartości. W podprocedurze `setUpGUI` wprowadź poniższą zmianę oznaczającą użycie wartości z zakresu ±500:

```
! Ustawienie zakresu wykresu i domeny. To trzeba zrobić
! po dodaniu pierwszego obiektu PlotPoint, ponieważ ten krok
! również powoduje ustawienie zakresu i domeny.
p.setView(-10, -500, 0, 500, 0)
```

Tytuł wykresu również należy zmienić, aby wskazywał użycie nowych jednostek pomiaru. Odpowiedni wiersz także znajduje się w podprocedurze `setUpGUI`:

```
! Inicjalizacja wykresu i jego wyświetlenie.
p = Graphics.newPlot
p.setTitle("Rotation in Degrees per Second")
```

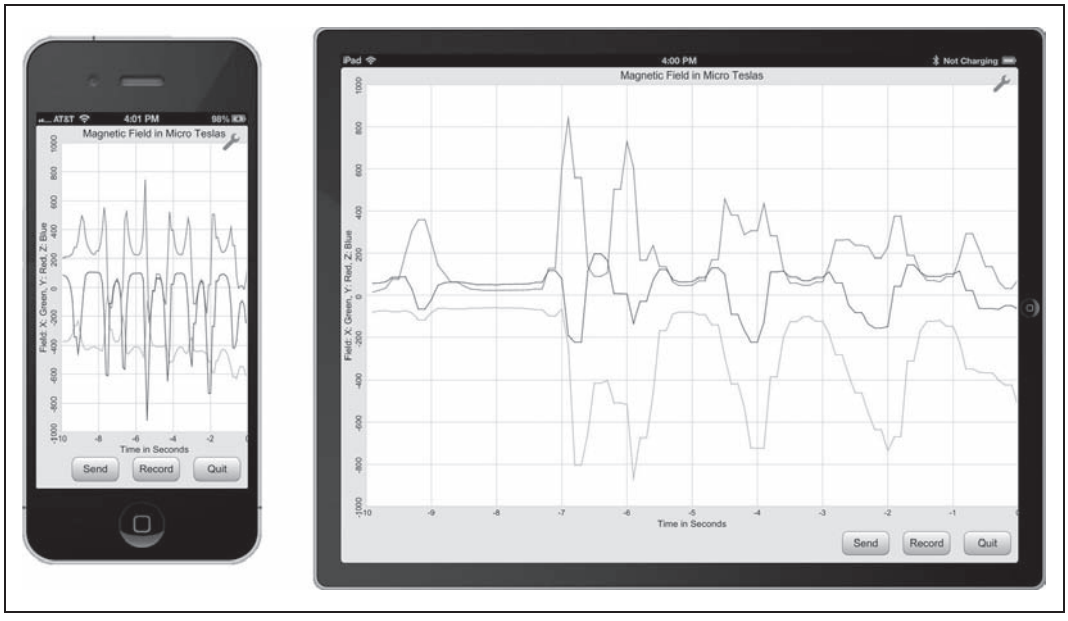
Magnetometr

Począwszy od modelu iPhone 3GS, wszystkie smartfony iPhone są wyposażone w magnetometr trójosiowy. Wymieniony czujnik jest wykorzystywany przede wszystkim jako kompas cyfrowy w aplikacjach opartych na mapach, choć potrafi zmierzyć także pole magnetyczne obecne w pobliżu urządzenia. Tę drugą możliwość wykorzystamy w rozdziale 3. do zbudowania prostego wykrywacza metalu. Natomiast w tym rozdziale utworzymy bardzo prostą aplikację pokazaną na rysunku 2.3. Magnetometru można użyć również do wyszukania przewodów oraz innych źródeł prądu lub po prostu można pobawić się za pomocą urządzenia iOS i magnesów przyczepianych do lodówki.

Oprogramowanie pobierające dane z magnetometru jest niemalże takie samo jak pobierające wartości z żyroskopu. Jak się domyślasz, między nimi istnieją pewne oczywiste różnice, na przykład odczyt danych z zupełnie innego czujnika, ale sama zasada działania jest podobna jak w przypadku programów odczytujących wartości z przyśpieszeniomierza i żyroskopu. W tabeli 2.1 wymieniono odpowiadające sobie polecenia w poszczególnych programach.

Poza tym istnieje jeszcze wiele innych oczywistych zmian kosmetycznych, takich jak dostosowanie nazw zmiennych, komentarzy i ciągów tekstowych, tak aby dotyczyły magnetometru.

Jedyna większa zmiana jest związana z ogromnymi wahaniami siły oddziaływania pól magnetycznych. Wartości maksymalne odczytywane z przyśpieszeniomierza sięgają ±2G we wszystkich urządzeniach iOS poza iPhone'em 5, w którym maksymalny zakres wynosi ±8G. Zdefiniowanie skali pionowej jako ±2G sprawdza się doskonale. W przypadku żyroskopu odczytywane wartości rzadko będą przekraczać 10 radianów na sekundę, a tym samym zakres ±10 dla osi Y w aplikacji Gyroscope sprawdza się doskonale. Jednak magnetometr jest bardzo czuły i może zmierzyć siłę oddziaływania ziemskiego pola magnetycznego. Siła wspomnianego pola magnetycznego nie jest stała i ogólnie zawiera się w przedziale od 30 do 60 mikrotesli (30–60 μT).



Rysunek 2.3. Uruchomiona aplikacja Magnetometer

Tabela 2.1. Odpowiadające sobie polecenia w poszczególnych programach

Accelerometer	Gyroscope	Magnetometer
`Sensors.accel`	`Sensors.gyro`	`Sensors.mag`
`Sensors.accelAvailable`	`Sensors.gyroAvailable`	`Sensors.magAvailable`
`Sensors.setAccelRate`	`Sensors.setGyroRate`	`Sensors.setMagRate`

Czujnik może wykryć znacznie silniejsze pola magnetyczne, maksymalnie do 1T, czyli 1000000 µT. Obsługa wartości w takim zakresie jest sporym wyzwaniem. Poniżej przedstawiono uaktualnioną wersję podprocedury `nullEvent`, która potrafi obsłużyć tak duży zakres wartości:

```
! Ta podprocedura jest wywoływana, gdy nic innego się nie dzieje.
! Sprawdza, czy od ostatniego odczytania wartości magnetometru
! upłynęło już co najmniej 0,1 sekundy. Jeśli tak, następuje wówczas
! pobranie i wyświetlenie nowych danych.
!
! Parametry:
!    time - godzina wystąpienia zdarzenia.
SUB nullEvent (time AS DOUBLE)

! Pobranie nowych wartości.
m = Sensors.mag

! Jeżeli zachodzi potrzeba, wartość zostaje zapisana w pliku danych wyjściowych.
IF recording AND (t0 <> m(4)) THEN
  PRINT #1, m(1); ","; m(2); ","; m(3); ","; m(4)
END IF

! Uaktualnienie wykresu.
IF m(4) > t0 + 0.1 THEN
   ! Uaktualnienie tablic przechowujących punkty tworzące wykres.
  WHILE m(4) > t0 + 0.1
```

```
      t0 = t0 + 0.1
      FOR i = 1 TO 99
        mx(i, 2) = mx(i + 1, 2)
        my(i, 2) = my(i + 1, 2)
        mz(i, 2) = mz(i + 1, 2)
      NEXT
      mx(100, 2) = m(1)
      my(100, 2) = m(2)
      mz(100, 2) = m(3)
    WEND

    ! Dostosowanie zakresu funkcji na podstawie maksymalnej zarejestrowanej wartości.
    max = 0
    FOR i = 1 TO 100
      IF ABS(mx(i, 2)) > max THEN max = ABS(mx(i, 2))
      IF ABS(my(i, 2)) > max THEN max = ABS(my(i, 2))
      IF ABS(mz(i, 2)) > max THEN max = ABS(mz(i, 2))
    NEXT
    range = 10^(INT(LOG(max)/LOG(10)) + 1)
    p.setView(-10, -range, 0, range, 0)

    ! Uaktualnienie wykresów.
    px.setPoints(mx)
    py.setPoints(my)
    pz.setPoints(mz)
  END IF
  END SUB
```

Zmiana znajduje się na końcu podprocedury, gdzie następuje dynamiczne dostosowanie zakresu. Program analizuje wartości, wyszukując największą. Następnie wykorzystuje pewną sztuczkę matematyczną, to znaczy na podstawie liczby całkowitej z logarytmu dziesiętnego dla największej wartości oblicza prawidłową potęgę dla zakresu pionowego. Przekonajmy się, jak takie rozwiązanie działa.

Zaczynamy od wartości odpowiadającej sile oddziaływania ziemskiego pola magnetycznego, czyli 50 µT. W takim przypadku wynik działania `LOG(50)/LOG(10)` wynosi nieco poniżej 1,7. Wyodrębniamy z niego liczbę całkowitą i dodajemy jeden, otrzymując tym samym wartość 2 — to jest liczba zer, które muszą się znaleźć w zakresie. Podnosimy liczbę 10 do obliczonej potęgi (2) i otrzymujemy zakres ±100, doskonale sprawdzający się podczas wyświetlania wartości od 10 do 100. Wypróbuj kilka innych liczb, a przekonasz się, że zastosowany przez nas wzór zawsze zwraca w wyniku 100 dla zakresu, gdy `max` wynosi więcej niż 10, ale mniej niż 100.

W pobliżu urządzenia iPhone lub iPad umieść źródło względnie silnego pola magnetycznego, a siła takiego pola wzrośnie do kilkuset µT. Spróbuj ponownie wykonać działania matematyczne, a przekonasz się, że dla wartości `max` od 100 do 1000 zakresem będzie ±1000.

To niesamowity widok, gdy aplikacja automatycznie dostosowuje pionowy zakres wykresu w odpowiedzi na zmianę siły pola magnetycznego. Z tym wiąże się jednak pewna wada zastosowanego rozwiązania: ręczne dostosowanie zakresu nie jest możliwe. Wprawdzie można to zrobić, ale program automatycznie dostosuje zakres po pobraniu następnej wartości z czujnika. Jeśli takie rozwiązanie Ci nie odpowiada, to usuń kod, który automatycznie ustala zakres.

Poniżej przedstawiono pełny kod źródłowy aplikacji odczytującej dane z magnetometru. Ten program o nazwie Magnetometer znajdziesz również w katalogu *O'Reilly Books* w aplikacjach techBASIC i techBASIC Sampler:

```
! Program wyświetla wykresy wartości pobranych z magnetometru w ciągu
! ostatnich 10 sekund w odstępach co 0,1 sekundy. Program oferuje funkcję rejestracji
! pobieranych wartości i wysłania wyników za pomocą poczty elektronicznej.

! Utworzenie wykresów i tablic przechowujących punkty wykresów.
DIM p as Plot, px as PlotPoint, py as PlotPoint, pz as PlotPoint
DIM mx(100, 2), my(100, 2), mz(100, 2)

! Utworzenie kontrolek.
DIM quit AS Button, record AS Button, send AS Button

! Utworzenie i inicjalizacja zmiennych globalnych używanych do śledzenia przebiegu działania programu.
fileName$ = "tempdata.txt"
recording = 0

! Pobranie i ustawienie wartości początkowej czasu dla magnetometru.
DIM t0 AS DOUBLE
IF Sensors.magAvailable THEN
  WHILE t0 = 0
    m = Sensors.mag
    t0 = m(4)
  WEND
END IF

! Utworzenie interfejsu użytkownika.
setUpGUI

! Utworzenie nowego przycisku i wypełnienie go gradientem.
!
! Parametry:
!   x - położenie poziome.
!   y - położenie pionowe.
!   title - tekst wyświetlany przez przycisk.
!
! Wartość zwrotna: nowy przycisk.
FUNCTION newButton (x, y, title AS STRING) AS Button
DIM b AS Button
b = Graphics.newButton(x, y)
b.setTitle(title)
b.setBackgroundColor(1, 1, 1)
b.setGradientColor(0.6, 0.6, 0.6)
newButton = b
END FUNCTION

! Ta podprocedura jest wywoływana, gdy nic innego się nie dzieje.
! Sprawdza, czy od ostatniego odczytania wartości magnetometru
! upłynęło już co najmniej 0,1 sekundy. Jeśli tak, następuje wówczas
! pobranie i wyświetlenie nowych danych.
!
! Parametry:
!   time - godzina wystąpienia zdarzenia.
SUB nullEvent (time AS DOUBLE)

! Pobranie nowych wartości.
m = Sensors.mag

! Jeżeli zachodzi potrzeba, wartość zostaje zapisana w pliku danych wyjściowych.
IF recording AND (t0 <> m(4)) THEN
  PRINT #1, m(1); ","; m(2); ","; m(3); ","; m(4)
END IF

! Uaktualnienie wykresu.
IF m(4) > t0 + 0.1 THEN
   ! Uaktualnienie tablic przechowujących punkty tworzące wykres.
```

```
  WHILE m(4) > t0 + 0.1
    t0 = t0 + 0.1
    FOR i = 1 TO 99
      mx(i, 2) = mx(i + 1, 2)
      my(i, 2) = my(i + 1, 2)
      mz(i, 2) = mz(i + 1, 2)
    NEXT
    mx(100, 2) = m(1)
    my(100, 2) = m(2)
    mz(100, 2) = m(3)
  WEND

  ! Dostosowanie zakresu funkcji na podstawie maksymalnej zarejestrowanej wartości.
  max = 0
  FOR i = 1 TO 100
    IF ABS(mx(i, 2)) > max THEN max = ABS(mx(i, 2))
    IF ABS(my(i, 2)) > max THEN max = ABS(my(i, 2))
    IF ABS(mz(i, 2)) > max THEN max = ABS(mz(i, 2))
  NEXT
  range = 10^(INT(LOG(max)/LOG(10)) + 1)
  p.setView(-10, -range, 0, range, 0)

  ! Uaktualnienie wykresów.
  px.setPoints(mx)
  py.setPoints(my)
  pz.setPoints(mz)
END IF
END SUB

! Ostatnio zarejestrowane dane będą wysłane za pomocą wiadomości e-mail.
SUB sendData
DIM e AS eMail
e = System.newEMail
IF e.canSendMail THEN
  e.setSubject("Magnetometer data")
  e.setMessage("Magnetometer data")
  e.addAttachment(fileName$, "text/plain")
  e.send
ELSE
  button = Graphics.showAlert("Can't Send",
     "Email cannot be sent from this device.")
END IF
END SUB

! Konfiguracja interfejsu użytkownika.
SUB setUpGUI

! Pobieranie wartości z magnetometru co 0,05 sekundy.
Sensors.setMagRate(0.05)

! Inicjalizacja tablic używanych przez wykres.
FOR t = 1 TO 100
  mx(t, 1) = t/10.0 - 10
  my(t, 1) = t/10.0 - 10
  mz(t, 1) = t/10.0 - 10
NEXT

! Inicjalizacja wykresu i jego wyświetlenie.
p = Graphics.newPlot
p.setTitle("Magnetic Field in Micro Teslas")
p.setXAxisLabel("Time in Seconds")
p.setYAxisLabel("Field: X: Green, Y: Red, Z: Blue")
p.showGrid(1)
```

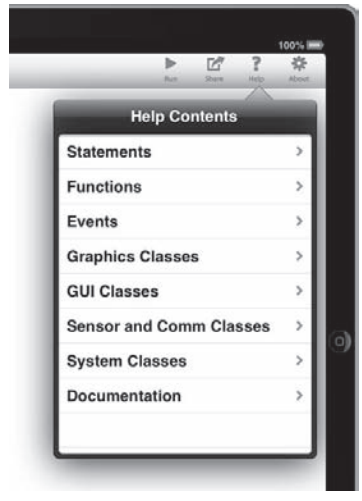
```
p.setGridColor(0.8, 0.8, 0.8)
p.setAllowedGestures($0042)

px = p.newPlot(mx)
px.setColor(0, 1, 0)
px.setPointColor(0, 1, 0)

py = p.newPlot(my)
py.setColor(1, 0, 0)
py.setPointColor(1, 0, 0)

pz = p.newPlot(mz)
pz.setColor(0, 0, 1)
pz.setPointColor(0, 0, 1)

! Ustawienie zakresu wykresu i domeny. To trzeba zrobić
! po dodaniu pierwszego obiektu PlotPoint, ponieważ ten krok
! również powoduje ustawienie zakresu i domeny.
p.setView(-10, -10, 0, 10, 0)

! Wyświetlenie widoku przedstawiającego wykresy. Przekazanie wartości 1
! dla parametru powoduje przejście do trybu pełnego ekranu.
system.showGraphics(1)

! Zablokowanie ekranu w jego bieżącej orientacji.
orientation = 1 << (System.orientation - 1)
System.setAllowedOrientations(orientation)

! Określenie wymiarów wykresu.
p.setRect(0, 0, Graphics.width, Graphics.height - 47)

! Wyświetlenie tła.
Graphics.setPixelGraphics(0)
Graphics.setColor(0.886, 0.886, 0.886)
Graphics.fillRect(0, 0, Graphics.width, Graphics.height)

! Konfiguracja interfejsu użytkownika.
h = Graphics.height - 47
quit = newButton(Graphics.width - 82, h, "Quit")
record = newButton(Graphics.width - 174, h, "Record")
send = newButton(Graphics.width - 266, h, "Send")

! Jeżeli nie ma żadnych danych do wysłania, wtedy należy wyłączyć przycisk Send.
IF NOT EXISTS(fileName$) THEN
  send.setEnabled(0)
END IF

! Upewnienie się o dostępności magnetometru. Jeżeli czujnik jest niedostępny,
! należy wyświetlić odpowiedni komunikat i zakończyć działanie programu.
IF NOT Sensors.magAvailable THEN
  msg$ = "This device does not have a magnetometer. "
  msg$ = msg$ & "The program will exit."
  button = Graphics.showAlert("No Magnetometer", msg$)
  STOP
END IF
END SUB

! Podprocedura wywoływana, gdy program powinien rozpocząć
! rejestrację danych. Ta podprocedura zmienia nazwę przycisku
! na Stop, otwiera plik danych wyjściowych, a następnie ustawia
! flagę wskazującą na konieczność rejestracji danych pobieranych
! z magnetometru.
SUB startRecording
```

```
record.setTitle("Stop")
recording = 1
OPEN fileName$ FOR OUTPUT AS #1
END SUB

! Podprocedura wywoływana, gdy program powinien zakończyć
! rejestrację danych. Ta podprocedura zmienia nazwę przycisku
! na Record, zeruje wartość zmiennej recording, a następnie zamyka
! plik danych wyjściowych.
!
! Tę podprocedurę można bezpiecznie wywołać nawet
! jeśli nie zostały zarejestrowane jakiekolwiek dane.
SUB stopRecording
IF recording THEN
  record.setTitle("Record")
  CLOSE #1
  recording = 0
  send.setEnabled(1)
END IF
END SUB

! Obsługa naciśnięcia dowolnego przycisku.
!
! Parametry:
!   ctrl - przycisk, który został naciśnięty.
!   time - godzina, o której wystąpiło zdarzenie.
SUB touchUpInside (ctrl AS Button, time AS DOUBLE)
IF ctrl = quit THEN
  stopRecording
  STOP
ELSE IF ctrl = record THEN
  IF recording THEN
    stopRecording
  ELSE
    startRecording
  END IF
ELSE IF ctrl = send THEN
  stopRecording
  sendData
END IF
END SUB
```

Uzyskanie szybszej odpowiedzi z czujnika

Zaprezentowane dotąd trzy programy pokazują, jak w zabawny i użyteczny sposób można pobrać niezmodyfikowane wartości z czujników przyśpieszeniomierza, żyroskopu i magnetometru. Wspomniane programy mają jednak pewną wadę. Ponieważ uaktualnienie środowiska graficznego zabiera dużą ilość czasu, częstotliwość pobierania wartości z czujników jest dość ograniczona. Wprawdzie istnieje możliwość pozbycia się grafiki i jedynie odczytywania danych z czujników, ale takie rozwiązanie, choć niewątpliwie zapewnia lepszą wydajność, również może wiązać się z utratą pewnych danych. Język techBASIC oferuje jeszcze inny sposób pobierania danych z czujników — wspomniane rozwiązanie pozwala na skrócenie czasu udzielania odpowiedzi przez czujniki. Pobieranie danych będzie odbywało się z maksymalną szybkością obsługiwaną przez iPhone, ale koszt takiego rozwiązania jest ogromny. Po wykonaniu zapytania program nie reaguje na działania użytkownika aż do chwili zebrania wszystkich danych z czujnika. Przeanalizujmy tego rodzaju rozwiązanie i przy okazji przekonajmy się, jakie są inne źródła pozwalające na jeszcze lepsze poznanie języka techBASIC.

techBASIC jest dostarczany wraz z wbudowanym systemem pomocy zawierającym informacje techniczne o każdym poleceniu, funkcji i klasie. Jeśli korzystasz z iPada, to naciśnij przycisk *Source* i spójrz na kod źródłowy programu, a następnie naciśnij przycisk *Help* znajdujący się na pasku narzędziowym. Z kolei w iPhone'ie musisz przejść do edycji programu, aby przycisk *Help* stał się widoczny. Po naciśnięciu przycisku *Help* na ekranie zostanie wyświetlona lista tematów pomocy pogrupowanych w odpowiednie kategorie (patrz rysunek 2.4).



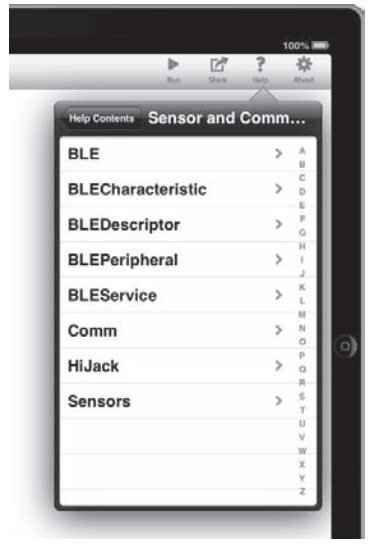
Rysunek 2.4. System pomocy wbudowany w aplikację techBASIC

Omówienie poleceń programu, na przykład polecenia `PRINT` użytego w pierwszym programie przedstawionym w tej książce, znajdziesz w kategorii *Statements*. Omówienie funkcji, na przykład `DEG` przeznaczonej do konwersji radianów na stopnie, znajdziesz w kategorii *Functions*. Z kolei wszystkie podprocedury wymieniono w kategorii *Events*, w której znajdziesz omówienie podprocedur specjalnych, takich jak `nullEvent` i `touchUpInside` wywoływanych przez techBASIC w programach opartych na zdarzeniach. W pozostałych kategoriach pogrupowano predefiniowane klasy.

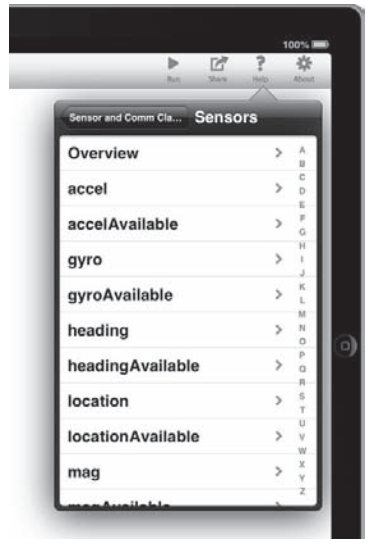
Chcemy bliżej poznać klasę `Sensors` omówioną w kategorii *Sensor and Comm Classes* pokazanej na rysunku 2.5. Naciśnij tę nazwę. Zobaczysz nazwy różnych klas pozwalających na uzyskanie dostępu do czujników zarówno wewnętrznych, jak i zewnętrznych. Te klasy będziemy intensywnie wykorzystywać w książce, więc warto zapoznać się z poświęconą im sekcją systemu pomocy.

Ostatnia klasa wymieniona na liście to `Sensors`, której używaliśmy w przykładach przedstawionych w tym rozdziale. Naciśnij jej nazwę. Opisy poszczególnych klas rozpoczynają się od ogólnego przedstawienia danej klasy (patrz rysunek 2.6). Zauważysz także kilka metod, które powinny być Ci już znane. My jednak szukamy nowej. Przewiń listę w dół i naciśnij metodę o nazwie `sample`.

Na rysunku 2.7 pokazano okno zawierające pełny opis metody `sample`. Potrafi ona jednocześnie odczytywać wartości z wszystkich trzech czujników, co pozwala na pobieranie z nich danych z maksymalną szybkością dozwoloną przez system operacyjny. Wspomniany opis zawiera wszystkie szczegółowe informacje techniczne o wywołaniu. Znajdziesz tutaj wszystko, czego potrzebujesz do utworzenia programu używającego danego wywołania.

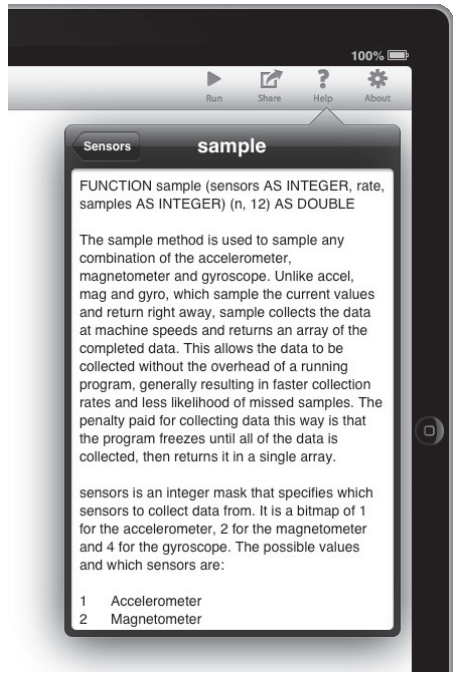


Rysunek 2.5. Kategoria Sensor and Comm Classes w systemie pomocy



Rysunek 2.6. Tematy pomocy dla klasy Sensors

W tym miejscu warto wspomnieć o źródle, które oferuje jeszcze więcej. System pomocy został zaprojektowany w taki sposób, aby można było szybko znaleźć potrzebne informacje i wyświetlić je na niewielkim ekranie. Nie zawiera zbyt wielu przykładowych kodów, a także jest pobawiony rysunków. Na szczęście dostępny jest podręcznik użytkownika techBASIC, w którym znajdziesz przykładowe fragmenty kodu i rysunki.



Rysunek 2.7. Opis metody sample

Podręcznik użytkownika dla techBASIC jest dostępny bezpłatnie na stronie Byte Works pod adresem *http://www.byteworks.us/Byte_Works/Products.html*. To dokument w formacie PDF, co pozwala na jego czytanie za pomocą aplikacji iBooks w urządzeniach iPad i iPhone.

Przeglądając fragment poświęcony metodzie `sample`, znajdziemy przedstawiony poniżej fragment kodu. Podobnie jak większość fragmentów kodu w podręczniku użytkownika to jest pełny, funkcjonujący program:

```
! Pobieranie danych z czujników przez 5 sekund.
samples = 250
samp = Sensors.sample(7, 0.02, samples)

! Określenie wartości średniej i maksymalnej
! dla każdego czujnika.
FOR i = 1 TO samples
  ax = ax + samp(i, 1)
  ay = ay + samp(i, 2)
  az = az + samp(i, 3)

  IF ABS(max) < ABS(samp(i, 1)) THEN max = samp(i, 1)
  IF ABS(may) < ABS(samp(i, 2)) THEN may = samp(i, 2)
  IF ABS(maz) < ABS(samp(i, 3)) THEN maz = samp(i, 3)

  gx = gx + samp(i, 5)
  gy = gy + samp(i, 6)
  gz = gz + samp(i, 7)
```

```
    IF ABS(mgx) < ABS(samp(i, 5)) THEN mgx = samp(i, 5)
    IF ABS(mgy) < ABS(samp(i, 6)) THEN mgy = samp(i, 6)
    IF ABS(mgz) < ABS(samp(i, 7)) THEN mgz = samp(i, 7)

    mx = mx + samp(i, 9)
    my = my + samp(i, 10)
    mz = mz + samp(i, 11)

    IF ABS(mmx) < ABS(samp(i, 9)) THEN mmx = samp(i, 9)
    IF ABS(mmy) < ABS(samp(i, 10)) THEN mmy = samp(i, 10)
    IF ABS(mmz) < ABS(samp(i, 11)) THEN mmz = samp(i, 11)
  NEXT

  PRINT USING "Max acceleration: ##.##, ##.##, ##.##"; max, may, maz
  PRINT USING "Max rotation: ###.##, ###.##, ###.##"; mgx, mgy, mgz
  PRINT USING "Max magnetic field: ###.##, ###.##, ###.##"; mmx, mmy, mmz

  PRINT USING "Average acceleration: ##.##, ##.##, ##.##"; _
              ax/samples, ay/samples, az/samples
  PRINT USING "Average rotation: ###.##, ###.##, ###.##"; _
              gx/samples, gy/samples, gz/samples
  PRINT USING "Average magnetic field: ###.##, ###.##, ###.##"; _
              mx/samples, my/samples, mz/samples
```

Choć niezbyt ładny wizualnie, to jest kompletny program pobierający z trzech omówionych dotąd czujników wartości z częstotliwością 50 razy na sekundę. Dane są zbierane przez pięć sekund. Następnie program oblicza i wyświetla wartości średnie i maksymalne.

Ten program nie jest jednym z przykładów zawartych w aplikacji techBASIC. W jaki sposób można go wypróbować w techBASIC? Niestety, firma Apple nałożyła przesadnie ogromne restrykcje mające na celu ochronę użytkowników iOS. Dlatego choć wydaje się to głupie, istnieją tylko dwa sposoby na wypróbowanie wspomnianego programu w techBASIC. Pierwszy wymaga jego samodzielnego wpisania. Drugi polega na wysłaniu kodu źródłowego w wiadomości e-mail, a następnie skopiowaniu tego kodu do nowego, pustego programu. Mimo wszystko użycie poczty elektronicznej nie jest najgorszym rozwiązaniem.

Wypróbuj ten program i przekonaj się, jak działa. Masz już wystarczająco dużą wiedzę, aby go zmodyfikować i rozbudować, jeśli zajdzie potrzeba.

Wyznaczanie kursu

W klasie `Sensor` po wyświetleniu opisu metody `sample` być może dostrzegłeś dwie inne usługi. Pierwsza, o nazwie `heading`, jest tak naprawdę zbiorem przetworzonych informacji pobranych z magnetometru, przyśpieszeniomierza i GPS. Zadanie, jakim jest wyszukanie kierunku na podstawie wspomnianych informacji, nie należy do łatwych, o czym się przekonasz w kolejnym rozdziale. Teraz wystarczy wiedzieć, że wyznaczenie kursu jest całkiem łatwe. Podobnie jak w przypadku trzech używanych dotąd czujników wszystko sprowadza się do odczytania wartości, prawdopodobnie po użyciu wywołania `headingAvailable` pozwalającego na sprawdzenie dostępności omawianej usługi w danym urządzeniu iOS. Z tym wiąże się jednak pewien problem. Pierwsze wywołanie `heading` włącza usługę, co wymaga chwili czasu. Zamiast wstrzymać działanie programu na czas włączania usługi, techBASIC zwraca ostatnio pobraną wartość. Należy pobrać kilka wartości, a nie tylko jedną:

```
FOR I = 1 TO 10
  PRINT Sensors.heading
  System.wait(0.5)
NEXT
```

Ten program dziesięciokrotnie odczytuje wartości, czekając pół sekundy między kolejnymi odczytami. Po jego uruchomieniu w moim iPadzie otrzymałem następujące wartości (puste wiersze zostały usunięte):

```
0             0             -1            22182.984556
90.786407     81.723602     -1            22183.473079
92.786407     83.723602     -1            22183.965387
38.786407     29.723602     40            22184.478982
34.786407     25.723602     40            22184.987621
34.786407     25.723602     40            22185.500116
33.786407     24.723602     40            22185.989068
33.786407     24.723602     40            22186.506593
33.786407     24.723602     40            22186.99292
33.786407     24.723602     40            22187.501888
```

Jak możesz się przekonać, wartości pochodzące z pierwszego odczytu są bezużyteczne.

Począwszy od wiersza drugiego, dwie pierwsze liczby to poprawne wartości kursu, podobnie jak w kompasie. Powyższe dane wyjściowe pokazują, że iPad był zwrócony w kierunku wschodnim, a następnie przed czwartym odczytem obrócony nieco w kierunku północno-wschodnim.

W rzeczywistości w ogóle nie dotykałem iPada. To tłumaczy potrzebę zastosowania trzeciej liczby w danych wyjściowych. Wspomniana liczba wskazuje dokładność odczytu, dla pierwszych trzech wynosi –1 i oznacza, że wartości w ogóle nie są dokładne, oraz przypomina nam o konieczności odczekania chwili, zanim urządzenie dostarczy prawidłowe dane. Nawet po osiągnięciu pełnej gotowości do pracy i podawania prawidłowych wartości dokładność nie jest zbyt dobra. System iOS wskazuje, że kurs jest podawany z dokładnością do 40 stopni.

Ostatnia wartość w danych wyjściowych wskazuje godzinę, o której nastąpił odczyt wartości.

Położenie

Ostatnia usługa w klasie `Sensors` to usługa podająca informacje o położeniu. Czy dane pochodzą z nowego czujnika, czy powstają na podstawie informacji pobranych z zewnętrznych czujników tworzących system satelitarnego GPS? Myślę, że to połączenie obu wspomnianych rozwiązań. W każdym bądź razie ta usługa jest używana podobnie jak usługa przeznaczona do wyznaczania kursu, a ponadto również potrzebuje nieco czasu, zanim osiągnie pełną gotowość do pracy i podawania prawidłowych wartości. Poniżej przedstawiono krótki program pokazujący, jak odczytywać informacje o położeniu. Ten fragment kodu został skopiowany bezpośrednio ze znajdującego się w podręczniku użytkownika techBASIC fragmentu prezentującego polecenie `location`:

```
location = sensors.location(30)
PRINT USING "Latitude    :  ####.###"; location(1)
PRINT USING "Longitude   :  ####.###"; location(2)
PRINT USING "Altitude    : #####.##"; location(3)
PRINT USING "Horiz. Error:  ####.###"; location(4)
PRINT USING "Vert. Error :  ####.###"; location(5)
PRINT USING "Speed       :  ####.###"; location(6)
PRINT USING "Direction   :  ####"; location(7)
PRINT       "Time stamp  : "; location(8)
```

Po uruchomieniu powyższego kodu w moim urządzeniu otrzymałem dane o położeniu, które okazały się całkiem dokładne! W poniższych danych wyjściowych usunąłem pewne liczby pozwalające na dokładne określenie mojego położenia.

```
Latitude    :    35.xxx
Longitude   :  -106.xxx
Altitude    :  1514.xx
Horiz. Error:    65.000
Vert. Error :    44.790
Speed       :    -1.000
Direction   :    -1
Time stamp  : 24355.097656
```

Zrozumienie pierwszych pięciu wartości nie powinno nastręczać trudności, jednostką miary dla odległości jest metr.

Szybkość (`Speed`) i kierunek (`Direction`) są obliczane na podstawie serii informacji o położeniu. Jeżeli te wartości nie mogą być obliczone, jak ma to miejsce w omawianym przykładzie, ponieważ iPad leży nieruchomo na biurku, wtedy wyświetlane są wartości –1. Kierunek nie jest kursem według kompasu, ale rzeczywistym kierunkiem ruchu.

Na końcu danych wyjściowych znajduje się znacznik czasu wskazujący godzinę, o której pobrano dane wartości.

Twój własny tricorder

W ten sposób otrzymałeś własny tricorder. Za jego pomocą możesz wyświetlać, rejestrować oraz wysyłać przez pocztę e-mail dane dotyczące przyśpieszenia, obrotu i siły pola magnetycznego. Dowiedziałeś się również, jak dzięki kilku dodatkowym wywołaniom uzyskać dostęp do informacji o kursie, położeniu, kierunku ruchu oraz szybkości.

W rozdziale 3. wykorzystamy zdobytą dotąd wiedzę do opracowania wykrywacza metalu opartego na magnetometrze. Nieco później, w rozdziale 7., powrócimy do przyśpieszeniomierza, choć wtedy przyśpieszeniomierzem będzie zdalne urządzenie. Przekonasz się, jak na podstawie danych przyśpieszeniomierza określić szybkość i pokonaną odległość, a także jak użyć przyśpieszeniomierza do określenia pułapu dla modelu rakiety.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Ż

Programowanie

O'REILLY®

Projekty elektroniczne na iPhone i iPad

Niekonwencjonalne gadżety z technologią Arduino i techBASIC

Podstawowe funkcje telefonu, czyli dzwonienie i wysyłanie SMS-ów, nikomu już dziś nie wystarczają. Współczesne smartfony są wykorzystywane na wiele innych sposobów. Gry, przeglądanie ulubionych stron w internecie, aktywny udział w życiu portali społecznościowych — to tylko niektóre z nich. Przy odrobinie umiejętności możesz użyć Twojego telefonu także do niekonwencjonalnych działań: na przykład jako wykrywacza metali, barometru lub żyroskopu. To urządzenia, które możesz zbudować na podstawie Twojego iPada lub iPhone'a oraz kilku niedrogich urządzeń dodatkowych. Dzięki tej książce jest to naprawdę proste! W trakcie lektury poznasz język techBASIC, który pomoże Ci zbudować działający higrometr oraz przyśpieszeniomierz. Ponadto zdobędziesz wiedzę na temat technologii Bluetooth Low Energy oraz nauczysz się sterować zdalnie samochodem za pomocą urządzenia typu BLE i platformy Arduino. Książka ta jest doskonałą lekturą dla wszystkich pasjonatów elektroniki, którzy chcieliby maksymalnie wykorzystać potencjał swoich smartfonów i tabletów. Zaskocz swoich znajomych niesamowitym zastosowaniem telefonu!

Dzięki tej książce:

- poznasz język techBASIC
- uzyskasz dostęp do czujników wbudowanych w Twoje urządzenie
- zbudujesz żyroskop, barometr lub higrometr
- wykorzystasz w pełni możliwości telefonu iPhone i tabletu iPad

Odkryj nowe zastosowania dla Twoich urządzeń iPhone i iPad!

helion.pl
księgarnia internetowa

Nr katalogowy: 20397

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-8890-6

9 788324 688906

Cena 59,00 zł

Informatyka w najlepszym wydaniu