```
      // PhoneGap ist geladen.
      //
      function onDeviceReady() {
        document.addEventListener("offline", onOffline, false);
      }

      // Offline-Event verarbeiten
      //
      function onOffline() {}
    </script>
  </head>

  <body></body>

</html>
```

***Listing 4–6*** *Offline-Beispiel*

Das Beispiel zeigt die gleiche Vorgehensweise wie beim `online`-Event. Daher möchte ich hier auf weitere Ausführungen verzichten.

**Online-Status prüfen**

Das Überprüfen der Internetverbindung sollte immer implementiert werden. Viele App-Store-Betreiber wie z.B. Apple testen als Erstes, was passiert, wenn eine Anwendung keinen Netzwerkzugriff hat. Sollten Sie also eine Anwendung erstellen, die Daten aus dem Internet nutzt, denken Sie an den Offline-Fall und berücksichtigen Sie ihn.

Jetzt haben Sie die notwendigen Events kennengelernt, um eine App auf Ihrem Smartphone zu nutzen. Jetzt soll es richtig mit dem Hardwarezugriff losgehen. Fangen Sie dazu im nächsten Abschnitt mit dem Kamerazugriff an.

## 4.3 Die Kamera

Mittlerweile besitzt fast jedes Smartphone eine Kamera. Für die Nutzung dieser Hardware haben alle Hersteller eine eigne App im jeweiligen Betriebssystem vorgesehen, deren Zugriff in der Regel relativ einfach ist. PhoneGap sieht dafür auch nur eine Methode für den Zugriff vor.


***Abb. 4–1*** *Standard-Kamera-App des iPhone*

### 4.3.1 Ein kurzes Beispiel

Bevor wir mit den Einzelheiten des Kamerazugriffs arbeiten, möchte ich ein einfaches Anwendungsbeispiel zeigen.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Kamera-Beispiel</title>
    <script type="text/javascript" charset="utf-8"
    src="cordova-2.0.0.js"></script>
    <script type="text/javascript" charset="utf-8">
      // Warten, bis PhoneGap geladen ist
      //
      document.addEventListener("deviceready", onDeviceReady, false);

      // PhoneGap ist bereit.
      //
      function onDeviceReady() {
        capturePhoto();
      }
```

```
      function capturePhoto() {
        navigator.camera.getPicture(onSuccess, onFail, {
          quality: 50,
          destinationType: Camera.DestinationType.FILE_URI
        });

      }

      function onSuccess(imageURI) {
        var image = document.getElementById('myImage');
        image.src = imageURI;
      }

      function onFail(message) {
        alert('Fehler: ' + message);
      }
    </script>
  </head>

  <body>
    <button onclick="capturePhoto();">Foto erstellen</button>
    <br>
    <img id="myImage" src="" />
  </body>

</html>
```

***Listing 4–7*** *getPicture-Beispiel*

Mit diesem Code ist es möglich, ein Bild zu erfassen und es in ein Image-Tag der HTML-Seite einzubetten.

#### Verfügbare Methoden

- `navigator.camera.getPicture`

Hiermit wird auf dem jeweiligen Gerät die native Bilderfassung gestartet. Die Methode inklusive Aufruf erläutere ich Ihnen im nächsten Kapitel. Zunächst schauen wir uns die notwendigen Objekte für den Aufruf an.

### 4.3.2 Das Options-Objekt

Obwohl es nur einen Methodenaufruf für die Kamera gibt, sind die Möglichkeiten vielfältig. Themen wie das Bildausgabeformat oder auch die Größe können hier festgelegt werden. Dafür sorgt das `option`-Objekt, das mit übergeben wird. Im Folgenden möchte ich die Werte kurz erläutern.

- `quality`
  Hier wird ein Wert zwischen 1 und 100 erwartet. Er gibt an, wie stark das Bild komprimiert werden soll.

- `destinationType`
  Mittels dieser Option können Sie festlegen, ob Sie den Pfad des Bildes als Rückgabewert wünschen oder eine Repräsentation als base64-codierten String. Eine Rückgabe als Binärstrom ist nicht vorgesehen. Dafür müssten Sie den Bildpfad weiterverarbeiten und per Dateioperationen darauf zugreifen.

```
Camera.DestinationType = {
// base64 encoded String
 DATA_URL : 0,
// Bildpfad als  URI
 FILE_URI : 1
};
```

- `PictureSourceType`
  Besagt, welche Quelle für das Bild zu nutzen ist. Hierbei ist nicht nur die Kamera gemeint. Sie können auch auf Ihre Fotogalerie oder Albenauswahl zugreifen. Hierdurch können Sie den `getPicture`-Aufruf auch als Bildauswahl nutzen.

```
Camera.PictureSourceType = {
 PHOTOLIBRARY : 0,
 CAMERA : 1,
 SAVEDPHOTOALBUM : 2
};
```

- `allowEdit`
  Bei dieser Option wird der Bearbeitungsdialog nach der Aufnahme ermöglicht. Hier kann das Bild beschnitten und seine Größe verändert werden – alles, was der Hersteller des Smartphones anbietet. Möchten Sie diese Nachbearbeitung ermöglichen, müssen Sie die Option mittels des booleschen Wertes `true` konfigurieren.

- `encodingType`
  Hiermit wird das Bildformat für die Ausgabe beeinflusst. Dabei ist zwischen JPEG und PNG zu wählen.

```
Camera.EncodingType = {
 JPEG : 0,
 PNG : 1
};
```

- `MediaType`
  Ist für die Auswahl von Bildern oder Videoaufnahmen verantwortlich. Es ist zu beachten, dass damit nicht die Aufnahme beeinflusst wird. Sie können es als einen Filter auf Medientypen wie Bilder und/oder Videos nutzen; allerdings nur, wenn Sie mit der Anweisung `PictureSourceType` die Fotogalerie oder das Fotoalbum gewählt haben.

```
Camera.MediaType = {
// Nur Bilder
PICTURE: 0,
// Nur Video
VIDEO: 1,
// Alle MedienTypen
ALLMEDIA : 2
};
```

- `targetWidth`, `targetHeight`
  Stellt die Bildgröße in Pixeln bereit. Es skaliert dabei das Bild. Die Parameter sind gemeinsam zu nutzen.
- `correctOrientation`
  Rotiert das Bild zur korrekten Ausrichtung, die auch beim Aufnehmen getroffen wurde. Das heißt, wurde im Landschaftsmodus aufgenommen, wird das auch in der Porträtansicht richtig angezeigt.
- `saveToPhotoAlbum`
  Als letzte Option steht `saveToPhotoAlbum` zur Verfügung. Hiermit kann das aufgenommene Bild sofort im Fotoalbum gespeichert werden.

Durch diese Fülle von Optionen können Sie sehr genau definieren, was die Bilderfassung tun soll. Die praktische Anwendung dieser Möglichkeiten können Sie in den kommenden Beispielen in Abschnitt 4.3.3 sehen.

**Galerie und Fotoalbum**

Auf einem Android-Smartphone zeigen die Optionen `PHOTOLIBRARY` und `SAVEDPHOTOALBUM` auf denselben Ordner, da Android diese Trennung nicht unterstützt.

### 4.3.3 camera.getPicture – ein Foto machen

| iOS | Android | Windows Phone | BlackBerry | Symbian | WebOS | Bada |
|---|---|---|---|---|---|---|
| x | x | x | x | – | x | x |

Für die Verwendung des Kamerazugriffs ist der Aufruf

```
navigator.camera.getPicture(success, fail, { options});
```

notwendig. Hierbei werden drei Parameter übergeben. Im Erfolgsfall wird das `success`-Callback ausgeführt. Bei einem Fehler oder Abbruch wird das `fail`-Callback genutzt. Um Themen wie Bildqualität oder Nachbearbeitung zu konfigurieren, werden diese in einem Options-Objekt festgelegt. Damit bildet es den dritten Parameter. Dieser Aufruf wird dann die native Kamerasoftware des Smartphones starten. Nach der Bilderfassung wird entweder der Erfolgs- oder Fehlerfall eintre-

ten bzw. ausgeführt. Wichtig ist natürlich die Frage: »Wo ist dann das Bild?« Da Sie das Options-Objekt schon kennen, kennen Sie auch die Antwort: je nach Optionswahl entweder im Dateisystem des Smartphones oder als base64-String im Speicher des Telefons. Die jeweiligen Optionen sehen Sie im folgenden Beispiel.


***Abb. 4–2*** *Fotoerfassung mit dem iPhone*

#### Ein komplettes Beispiel

Als Beispiel soll uns eine einfache HTML-Seite dienen, die ausgewählte oder erfasste Bilder anzeigt.

Dabei wird das ausgewählte Bild in einem Image-Container angezeigt. Die vier Buttons verweisen dabei auf verschiedene Arten der Bilderfassung.

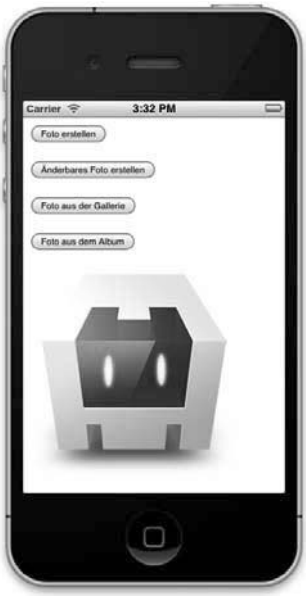


***Abb. 4–3*** *getPicture-Beispiel*

```
<!DOCTYPE html>
<html>

  <head>
    <title>Foto erfassen</title>
    <script type="text/javascript" charset="utf-8"
      src="cordova-2.0.0.js"></script>
    <script type="text/javascript" charset="utf-8">
      // Warten, bis PhoneGap geladen ist
      document.addEventListener("deviceready", onDeviceReady, false);

      var pictureSource; // Bild Quelle
      var destinationType; // Zielformat


      // PhoneGap ist bereit.
      function onDeviceReady() {
        pictureSource = navigator.camera.PictureSourceType;
        destinationType = navigator.camera.DestinationType;
      }

      // Wird aufgerufen, wenn ein Foto gemacht wurde.
      //
      function onPhotoDataSuccess(imageData) {
```

```
    // Bild-Platzhalter aus dem HTML bekommen
    //
    var smallImage = document.getElementById('smallImage');

    // CSS-Style ändern, um das Bild anzuzeigen
    //
    smallImage.style.display = 'block';

    // Das Bild anzeigen
    //
    smallImage.src = "" + imageData;
  }

  // Wird aufgerufen, wenn ein Foto gemacht wurde.
  //
  function onPhotoURISuccess(imageURI) {
    // Bild-Platzhalter aus dem HTML bekommen
    //
    var largeImage = document.getElementById('largeImage');

    // CSS-Style ändern, um das Bild anzuzeigen
    //
    largeImage.style.display = 'block';

    // Das Bild anzeigen
    //
    largeImage.src = imageURI;
  }


  function capturePhoto() {
    navigator.camera.getPicture(onPhotoDataSuccess, onFail, {
      quality: 50,
      destinationType: destinationType.DATA_URL
    });
  }

  function capturePhotoEdit() {
    navigator.camera.getPicture(onPhotoDataSuccess, onFail, {
      quality: 20,
      allowEdit: true,
      destinationType: destinationType.DATA_URL
    });
  }

  function getPhoto(source) {
    navigator.camera.getPicture(onPhotoURISuccess, onFail, {
      quality: 50,
      destinationType: destinationType.FILE_URI,
      sourceType: source
    });
  }
```

```
      function onFail(message) {
        alert('Fehler: ' + message);
      }
    </script>
  </head>

  <body>
    <button onclick="capturePhoto()">Foto erstellen</button>
    <br>
    <br>
    <button onclick="capturePhotoEdit()">Auml;nderbares Foto erstellen</button>
    <br>
    <br>
    <button onclick="getPhoto(pictureSource.PHOTOLIBRARY)">
      Foto aus der Galler-ie</button>
    <br>
    <br>
    <button onclick="getPhoto(pictureSource.SAVEDPHOTOALBUM)">
      Foto aus dem Al-bum</button>
    <br>
    <br>
    <img style="display:none;width:60px;height:60px;" id="smallImage"
      src=""
    />
    <img style="display:none;" id="largeImage" src="" />
  </body>

</html>
```

***Listing 4–8*** *getPhoto-Beispiel*

Auch hier möchte ich mit Ihnen das Listing Schritt für Schritt durchgehen. Zu Beginn wird wieder die PhoneGap-JavaScript-Datei eingebunden.

```
<script type="text/javascript" charset="utf-8"
 src="cordova-2.0.0.js"></script>
```

Darauf folgt das obligatorische Registrieren des `onReady`-Events.

```
document.addEventListener("deviceready", onDeviceReady, false);
```

Anschließend werden zwei Variablen deklariert, die wir im weiteren Verlauf noch nutzen werden.

```
var pictureSource;
var destinationType;
```

Wenn das `onReady`-Event eintrifft, werden die beiden zuvor erstellten Variablen auch belegt. Die Variablen nehmen dann die möglichen Werte der Optionen `pictureSource` und `destinationType` an.

```
function onDeviceReady() {
    pictureSource=navigator.camera.PictureSourceType;
    destinationType=navigator.camera.DestinationType;
}
```

Alternativ hätte man auch einfach die jeweiligen Werte an das Options-Objekt übergeben können. Aber sollten sich die Werte einmal ändern, sind diese Werte nicht hart codiert. Daher wird hier mit den Namen gearbeitet.

Als Nächstes folgt eine Funktion, die für das weitere Verarbeiten von Bildaufnahmen gedacht ist.

```
function onPhotoDataSuccess(imageData) {
    var smallImage = document.getElementById('smallImage');
    smallImage.style.display = 'block';
    smallImage.src = "" + imageData;
}
```

Dabei bekommt die `onPhotoDataSuccess`-Funktion den Parameter `imgData` übergeben. Hier ist dann das Bild enthalten. In der Funktion selber werden nur zwei Dinge getan: Einmal wird die DOM-Funktion `getElementById` ein HTML-Element mit der `id='smallImage'` an die JavaScript-Variable `smallImage` übergeben. Danach folgt das Füllen der Variablen mit den Bilddaten. Da diese aktuell im Standard per base64-Codierung übergeben wird, setzen wir im Skript noch den MIME-Type auf JPEG im base64-Format.

Die folgende Funktion `onPhotoURISuccess` ist für das Verarbeiten von Bildern mittels Dateipfaden gedacht.

```
function onPhotoURISuccess(imageURI) {
    var largeImage = document.getElementById('largeImage');
    largeImage.style.display = 'block';
    largeImage.src = imageURI;
}
```

Hier gibt es demzufolge keine Bilddaten, sondern den Speicherort des Bildes als Pfadangabe `imageURI` als Parameter. Danach folgt ein ähnliches Verfahren wie bei der `onPhotoDataSuccess`-Funktion. Es wird wieder die DOM-Funktion `getElementById` genutzt, um ein HTML-Element mit der `id='largeImage'` zu finden und der JavaScript-Variablen `largeImage` zu übergeben. Es wird das `display` CSS-Attribut gesetzt, da der HTML-Teil bis zu diesem Zeitpunkt noch auf `hidden` steht. Danach folgt das Füllen der Variablen mit den Bilddaten. Der Unterschied ist hierbei, dass der Pfad zum Bild auf das `src`-Attribut des Image-Tags gesetzt wird. Diese Art der Anzeige ist wesentlich ressourcenschonender, da kein base64-String im Speicher gehalten werden muss.

Es folgen jetzt die jeweiligen Funktionen für die Bilderfassung bzw. den Zugriff auf die Fotogalerie. Ich möchte hier nur die Funktion `capturePhotoEdit` besprechen, da die zwei anderen nur Abwandlungen darstellen.

```
function capturePhotoEdit() {
    navigator.camera.getPicture(onPhotoDataSuccess, onFail, {
      quality: 20,
      allowEdit:true,
      destinationType: destinationType.DATA_URL
    });
}
```

Diese Funktion ruft die `getPicture`-Methode von PhoneGap auf. Dabei wird im Erfolgsfall die Funktion `onPhotoDataSuccess` ausgeführt, im Fehlerfall dagegen `onFail`. Das Options-Objekt wird hier mit den Werten `quality:20`, `allowEdit:true` und `destinationType: destinationType.DATA_URL` übergeben. Es wird also die Qualität der Aufnahme festgelegt. Eine Nachbearbeitung im Aufnahmedialog ermöglicht `allowEdit:true`. Danach wird als Rückgabewert der Dateipfad festgelegt. Das passiert per `destinationType`. Dabei wird dieser allerdings nicht als Zahl mit dem Wert 1 übergeben, sondern per Variable. Damit ist bei einem Wechsel des Wertes in einer neuen Version keine Codeänderung nötig.

Als letzte Deklaration im JavaScript-Teil folgt noch die Fehlerbehandlung per `onFail`-Funktion. Hier ist diese sehr einfach gehalten und gibt die Fehlermeldung per Alert-Box aus.

```
function onFail(message) {
  alert('Fehler: ' + message);
}
```

Lassen Sie uns das Beispiel kurz mit einem Überblick des HTML-Teils abschließen. Es sind vier Button-Tags für die verschiedenen Aufrufe erstellt worden. Dabei wird der `onclick`-Handler mit der vorher definierten JavaScript-Funktion belegt.

```
<button onclick="capturePhoto()">Foto erstellen</button>
```

Als Platzhalter für die anzuzeigenden Bilder sind zwei Image-Elemente erstellt worden – jeweils mit einer ID versehen. Diese wird im JavaScript-Code für den Zugriff auf das jeweilige Element genutzt. Zu beachten ist hierbei, dass per `style`-Anweisung die Elemente nicht sichtbar sind, bis dies explizit im JavaScript-Code gesetzt wird.

```
<img style="display:none;" id="largeImage" src="" />
```

Mit dem oben gezeigten Beispiel haben Sie einen guten Überblick über die verschiedenen Einsatzmöglichkeiten der Kamera-API erhalten. Sie sind nun in der Lage, den Zugriff nebst weiterer Verarbeitung mit PhoneGap zu meistern.

Nach dem Zugriff auf die Kamera geht es mit dem beliebten Feature Geolocation, also dem Herausfinden der aktuellen Position, weiter.

**Ein Problem bei Windows Phone 7**

Wenn bei einer aktiven Verbindung per Zune-Software die native Kamera-App aufgerufen wird, wird dies nicht funktionieren und der Error-Callback wird ausgeführt.

## 4.4 Geolocation

Mittels Geolocation-API können Sie auf den GPS-Sensor Ihres Smartphones zugreifen. Dabei werden aber nicht nur die Daten des GPS-Sensors, sondern auch die Ortbestimmung per sog. Assisted-GPS[1] genutzt. Das heißt, die ungefähren Geodaten der Funkzelle oder die GPS-Daten des WLAN-Hot-Spots werden genutzt. Damit ist eine schnellere und exaktere Bestimmung möglich. Über die gesamten Interna müssen Sie sich dabei keine Sorgen machen. Es reichen bereits drei Methoden, um den Standardort des Nutzers abzufragen.

Wenn Sie schon mittels HTML5 auf Geodaten zugreifen, kennen Sie sich bereits etwas aus. PhoneGap hat sich bei den Aufrufen der API an die W3C-Spezifikation der Geolocation-API[2] gehalten. Sollten Sie ein Gerät haben, das diesen Standard bereits nutzt und damit Zugriff auf den GPS-Sensor gibt, wird automatisch die Geräteimplementierung genutzt. Einige Betriebssysteme implementieren die Gelocation-API aber nicht oder nur teilweise. Dies ist für Sie als Entwickler unproblematisch, da PhoneGap in solchen Fällen die Geolocation-Funktionen selbst bedient. Der Zugriff erfolgt dabei über folgende Methoden.

#### Verfügbare Methoden

- `geolocation.getCurrentPosition`
- `geolocation.watchPosition`
- `geolocation.clearWatch`

#### Verfügbare Objekte

Die Geolocation-API gibt Ihnen anstatt roher Daten lieber Objekte mit dem Ergebnis zurück. Es handelt sich dabei um drei Objekte.

#### Das Position-Objekt

Dieses Objekt kennt die Koordinaten und einen Zeitstempel. Es wird als Rückgabewert von Positionsanfragen genutzt. Die verfügbaren Eigenschaften sind:

1. *http://de.wikipedia.org/wiki/Assisted_Global_Positioning_System*
2. *http://dev.w3.org/geo/api/spec-source.html*