brief contents

1 ■ Barcodes, iOS, and you 1
2 ■ Media capture with AV Foundation 18
3 ■ Scanning barcodes 48
4 ■ Passbook, Apple’s digital wallet 70
5 ■ Generating barcodes 97
6 ■ Getting metadata for barcodes 133
7 ■ Putting barcodes in context 172

appendix A ■ History of the UPC 205
appendix B ■ GTIN prefix ranges 212
appendix C ■ GS1-128 application identifiers 217
Barcodes, iOS, and you

This chapter covers

- Why the nexus of barcodes and mobile technologies is creating new, exciting opportunities for app makers
- The barcode symbologies in iOS you should know about
- The distinctions between 1D and 2D barcodes
- A brief history of the UPC/GTIN, the mother of modern barcodes

In the past, if you wanted to add barcode scanning to your apps, you had to either fight your way through open source projects or license a commercial barcode-scanning library. None of those projects were written in Objective-C, documentation was lacking, and commercial solutions required payment of license fees for each downloaded copy of your app. All of these issues made barcode scanning impractical for all but the most skilled iOS developers, and too expensive to make economic sense for free or low-cost apps.

When Apple added Passbook to iOS 6, they built in the ability to display barcodes on Passbook passes. With iOS 7, Apple made these APIs public and added the
ability to scan barcodes. This allowed them to add barcode-scanning functionality to several of their first-party apps:

- The **Passbook app** lets you add new passes to your device by scanning special QR Codes.
- The **iTunes app** has the ability to redeem iTunes credits by scanning a voucher.
- The **Apple Store app** has an in-store UI that lets you scan the barcodes of accessories for unassisted checkout (see figure 1.1).

In June 2014, the mother of all barcodes, the UPC, celebrated its 40th anniversary. This makes it an incredibly well understood and ubiquitous technology. Throughout these four decades, different usage scenarios prompted the development of a variety of barcode symbologies that were more or less all informed by the UPC. Apple selected from these the most prevalent and useful kinds of barcodes to support in iOS 7. Support for a few additional barcode symbologies was added in iOS 8. To grasp the full potential of these differing symbologies—as they’re relevant to you in iOS app development—you’ll learn which purposes they’re best used for.

This introductory chapter will give you a solid understanding of barcode technology. Seeing how the multitude of symbologies relate to each other should alleviate any anxiety you might feel right now. You’ll no longer shiver in fear from not knowing the difference between UPC, EAN, GTIN, Code 25, Code 39, Code 93, and Code 128. A brief
history of the UPC/GTIN will prove illuminating. Not only has its long history been quite amusing at times, this background will aid greatly in your appreciation of the current state of the GTIN. You’ll become a barcode guru and be able to hold your own in any discussion about barcodes.

In order to appreciate the power of the barcode, we’ll first take a look at how they evolved. Beginning with the UPC, more and more barcode symbologies evolved over time because their predecessors had been designed to solve very specific problems. If you know how to tell them apart—just from glancing at them—you’ll know if you’re looking at an opportunity for a new app.

1.1 **The evolution of barcodes**

The first barcode in wide use was the *Universal Product Code* (UPC), combining the semantic meaning of a 12-digit number with a machine-readable scheme for representing this number as a series of bars. It was designed only for automated handling of physical products and was therefore limited to representing numerical product codes. Appendix A will walk you through the history of the UPC and how it became the GTIN, as it’s referred to nowadays by people in the know. Figures 1.2a and 1.2b show some examples of how barcodes have changed over the years.

**Figure 1.2a** **Timeline of barcodes**
Having overcome the hurdle of enabling a machine to recognize visual markings with a laser beam, a plethora of other kinds of barcodes started to appear, all with more-or-less specific fields of application. For example, the post office found that adding markings to mailed items would allow them to automatically sort the items. Luggage for airline travel was similarly tracked with numeric codes. Other industries had their own standards that worked better for them.

A combination of several bars that make up an individual character or digit is often called a symbol. The set of symbols available for a specific barcode standard is referred to as its symbology. All these different symbologies can be read with a laser beam.

### 1.1.1 One dimension: laser

Think of a laser beam as cutting out a horizontal slice of the vertical code bars. As the beam moves over the symbol (see figure 1.3), it measures the relative time it spends scanning dark bars and light spaces. A lookup table is then used to decode individual characters from those times.

The line of the laser beam is also the reason why these kinds of barcodes are referred to as being one-dimensional (1D).

If you have more-complex encoding schemes, you can also represent letters and special characters. Some 1D barcode types were created to represent short texts.

The long-recognized major advantage of 1D barcodes is that they can be decoded extremely reliably even when the items
tagged with such codes are moving at high speed. Some schemes even employ checksums to recognize when something is misread and increase this reliability.

The second advantage of 1D barcodes is cost. Because the technology has been around for 40 years now, the necessary components (laser diode and decoding electronics) have become cheap and reliable. This makes them ideally suited for high-volume deployment as well as for use in environments where you need to scan a great many codes in quick succession.

### 1.1.2 Two dimensions: CCD

The charge-coupled device (CCD) was invented at AT&T Bell Labs by Willard Boyle and George E. Smith in 1969. This is the chip at the heart of any kind of digital camera. Curiously, the technology for CCDs was invented around the time the first 1D barcode was introduced, but it took decades to develop CCDs to the point where they could compete with the accuracy of their technically much simpler predecessors.

A CCD is essentially a matrix of pixels that reads different binary values for each pixel depending on the light intensity shining on it. As a result, a CCD can read barcodes just like a laser beam can if you have a sufficient number of pixels (a.k.a. resolution). Because the CCD pixels are laid out in two dimensions, CCDs are also able to recognize a new kind of barcode, the two-dimensional (2D) barcode.

Freed of the limitations of one dimension, 2D barcodes usually consist of small rectangles laid out to form a square grid. Figure 1.4 shows such a barcode.

Initially CCDs didn’t have the necessary resolution, nor did CPUs have the decoding power, required for barcode recognition. A CPU essentially needs to look at each individual frame of video coming from the CCD and look for patterns that constitute a code. Significant advances in electronics and computer vision were necessary to be able to do that.

As with all computer technology, Moore’s Law worked its magic on CCDs to eliminate these hurdles over time. Modern smartphones have resolutions measured in megapixels, which are more than sufficient for scanning 1D and 2D barcodes.

There are only two reasons why laser-based scanners are still more widely used at supermarket checkouts than CCD-based scanners: First, CCD hardware is more complex and expensive than laser scanners. And second, CCDs were limited in how many frames per second of video they could capture. If you moved the scanned code too quickly, all a decoder saw were blurred images, and it would be unable to recognize any barcodes. Modern CCDs are capable of sufficiently fast shutter speeds that the blurriness of individual frames is much less of an issue.
1.1.3 Versatility is winning

Imagine setting up a competition for accuracy and speed between laser- and CCD-based scanners, of course using the latest and greatest models. The test scenario is a checkout stand at a supermarket. The contestants are professional cashiers with years of experience scanning products. Whoever scans 1,000 products first, distributed among 100 shopping carts each, will be the winner.

Who do you think will win? Laser or CCD?

I’d still bet on the laser winning this race because of its history and the fact that it’s been optimized for exactly these kinds of high-speed scenarios. But any relevant difference in technical ability or cost will continue to melt away in the coming years.

One disadvantage of laser-based scanners remains: they will never be able to scan 2D barcodes, whereas CCD scanners are more versatile. There are many scenarios where versatility trumps speed. Consider yourself and your cellphone: you’re never going to compete for speed with a professional cashier, but having the ability to scan barcodes might be a very welcome function in your pocket.

All modern smartphones have a camera built in for taking photos. With a bit of software intelligence, all those cameras gain the ability to scan barcodes at no extra cost.

1.1.4 Where are the bars?

You might wonder why I’m referring to the two-dimensional technology as “2D barcodes” even though there are no bars in sight. All 1D barcodes really do have bars representing a code; most 2D barcodes have small squares. It would be grasping at straws to argue that squares are really bars that are very fat and not very tall.

This technical inaccuracy is why the hyper-precise Apple engineers refer to these codes as machine-readable codes. But despite the logic of this term, nobody so far has suggested a term for 2D barcodes that has stuck and found wide usage.

Other terms you’ll sometimes hear—like “QR Code”—refer to a single member of the family of 2D barcodes and thus are just as inaccurate. If you wanted to be extra-precise, the correct technical description would be something like “machine-readable codes that use markings forming a matrix grid.”

You’ll probably agree that “2D barcode” is still the best option, however inaccurate it may be. It simply rolls off the tongue better.

1.2 Barcode symbologies in iOS

There are far too many types of 1D and 2D barcodes to cover all of them in this book. This book is about Apple’s support for barcodes, so we’ll only look at the types of barcodes that the operating system actually supports. Apple added general barcode support in iOS 7 and added support for a few more symbologies in iOS 8. Figure 1.5 gives a categorized overview of all supported symbologies in iOS.

We won’t go into the details of how the individual symbologies are constructed. This overview will give you the basics you need to understand the supported barcode types, their abilities, and where they’re primarily used.
## 1.2.1 1D barcodes in iOS

Barcodes are said to have one dimension if there’s a single line (such as a line traced by a scanner’s laser) that can cross all lines of the symbol. iPhones don’t have a built-in laser for scanning, but the single-line scanning can be emulated with the camera. The AV Foundation framework scans multiple horizontal and vertical lines on the images coming from the camera and recognizes a 1D barcode as soon as one scan line crosses all bars of the code.
THE GTIN FAMILY

The first commercial barcode, the Universal Product Code (UPC), was announced—after much deliberation—in a press release in April 1973. The first-ever product carrying a UPC code in its packaging was scanned in June 1974. It was a 10-pack of chewing gum, now on display at the Smithsonian in Washington, DC. If you’re interested in the curious history of how this came to be, please turn to appendix A, where I tell the whole story.

The GS1 organization (www.gs1.org) maintains the standards related to the Global Trade Item Number (GTIN). There are several symbologies that belong to this family, all of them representing a product code and all using the same kind of barcode symbols:

- **UPCA**—The classic first product barcode (12 digits). GS1 refers to it as GTIN-12.
- **UPCE**—A narrower version for compressed product numbers (8 digits).
- **EAN-13**—The classic European barcode with a thirteenth digit; in Japan it’s referred to as the Japanese Article Number (JAN) (13 digits). GS1 refers to it as GTIN-13.
- **EAN-8**—A narrower version of the EAN for compressed product numbers (8 digits). GS1 refers to it as GTIN-8.

These variations are illustrated in figure 1.6.

![Figure 1.6 The four members of the GTIN family of barcodes](image-url)
For the narrower symbologies, product numbers that contain several zeros in the middle can be compressed by suppressing the string of zeroes. This requires a manufacturer prefix that ends in zeroes (by special request from GS1) as well as a product number that’s prefixed with zeros. Manufacturers love to use these for products with little space on the packaging.

All product codes represented by barcodes of the GTIN family expand to 13-digit numbers. UPC-A (the “classic” code used in the USA) just gets an extra zero as a prefix. Uncompressing UPC-E and EAN-8 to 13 digits is slightly more complicated, but still possible.

NOTE Paper magazines often have an additional barcode next to the GTIN. This is used to encode the issue number with EAN-2 or the price with EAN-5. Those extra barcodes are currently not supported by iOS and are ignored when scanning the EAN-13.

Of these 13 digits, the rightmost is the check digit. After scanning, the check digit must match a calculation based on the other numbers; if it doesn’t match, the scan is considered corrupted.

The first digit is not directly encoded as bars, but is represented by a certain pattern of odd and even bars. Thus US GTINs always use the same pattern, tied to the prefix 0, whereas international GTINs have 9 other patterns to choose from. When scanning a barcode, the scanner infers the leading digit from the pattern of odd and even symbols.

The left half of GTINs contain the manufacturer prefix, which is assigned by GS1. The organization responsible for the manufacturer’s country assigns those prefixes based on the table reproduced in appendix A. For example, a company located in Taiwan would have its manufacturer prefix assigned by GS1 Taiwan, and it would have a 470 prefix.

After that, each manufacturer decides on the numbers they assign to their products. If they were to assign a very low number, with several leading zeroes, this would allow for the number compression in UPC-E and EAN-8.

The International Standard Book Number (ISBN) is a variant of the GTIN-13 used for books and book-like publications. In this case, the fixed prefix 978 or 979 replaces the manufacturer prefix, and the remaining digits are assigned by national ISBN agencies. ISBNs have their own check digit, which is calculated differently than the one in the GTIN.

The International Standard Music Number (ISMN) is a unique number for the identification of all notated music publications from all over the world. It uses a fixed prefix of 9790. ISMNs are also assigned by national agencies.

The third kind of special GTIN-based numbering scheme is the International Standard Serial Number (ISSN), used for printed or electronic periodical publications. They share the 977 prefix and are assigned by national centers of the ISSN organization, which is independent of GS1.
CHAPTER 1  Barcodes, iOS, and you

GTIN-12/UPC-A: Identifies the product at POS for checkout. You can identify this as a UPC because there are 12 digits shown.

Code 39: Encodes model identifier, hardware revision, and unique serial number. It simplifies adding this device to a corporation's hardware inventory.

Figure 1.7  Netgear combines a UPC with three Code 39s to simplify adding this router device to a corporate tech inventory-tracking system.

NOTE  There are several additional GTIN prefix ranges used for products that are packaged in stores, like cheese or meat. Also, you can infer the country that a product manufacturer is located in from these prefixes. For more details, refer to the table “GTIN-13 prefixes” in appendix B.

CODE 39 AND CODE 39 MOD 43
Code 39 was developed by Dr. David Allais and Ray Stevens of Intermec (now a subsidiary of Honeywell Scanning and Mobility) in 1974 to overcome the limitation of the just-released UPC-A, which was only able to represent digits. They saw the promise of automating identification and data capture, but wanted to be able to represent letters as well.

Code 39 is also known as Alpha39, Code 3 of 9, Code 3/9, Type 39, USS Code 39, and USD-3. Its name derives from it initially being able to represent 39 different characters (plus a symbol to represent start and stop)—generally only uppercase letters. With the help of control characters, it can also represent the full ASCII set of characters.

Another advantage that Code 39 has over GTIN is that it can be used for any length of text. But as the number of characters increases, so does the width of the barcode representation. Figure 1.7 shows several Code 39 examples that represent a device model number and its serial number in scannable form.

The plain version of Code 39 doesn’t have a checksum to detect scanning errors. A variant of Code 39 does have such a check digit, involving a modulo 43 operation; it’s referred to as Code 39 mod 43. iOS supports both variants.

A human-readable version of the represented text is usually displayed beneath the code, with an asterisk representing the start/stop character.

NOTE  Because Code 39 is as old as the UPC, virtually all existing barcode scanners are able to read Code 39 codes.

CODE 93
Code 93 is a descendant of Code 39, developed by Intermec eight years after its predecessor, in 1982. Its design goal was to reduce the horizontal space required to represent the text and improve error detection by adding a checksum. Figure 1.8 shows examples of the two codes.
The origins of the name are unknown—there’s nothing in the standard with the number 93. Maybe it’s just a play on the name of its predecessor. By placing the 9 in front, it sounds newer, bigger, and better. That’s marketing for you.

Code 93 at its core has 43 characters and 5 special characters. Like Code 39, it can represent all 128 ASCII characters with the help of control characters. This feature was tacked on to Code 39 by reusing codes, but in Code 93 dedicated control characters are used.

The main benefit of Code 93 is that if you scan such a code with iOS, you get the ASCII characters decoded, whereas with Code 39 you have to find the control sequences and do the decoding yourself.

**Code 128**

Code 128 was developed in 1981 by Computer Identics Corporation, and its name references the fact that it can represent the full 128 characters of the ASCII code.

Data density in Code 128 is comparable to Code 93, and the features are quite similar. Figure 1.9 shows a sticker from an Apple product box where you can see two Code 128 symbols representing supplementary information about the device.

Code 128 has a more sophisticated mechanism for extending the range of characters—it uses three different sets of character codes specified by control characters. Those character sets are referred to as 128A, 128B, and 128C.

Code 128 has a mechanism that allows it to save horizontal space. The 128C symbol set allows two neighboring digits to be encoded in one code symbol. This compression
CHAPTER 1  Barcodes, iOS, and you

Code 128:
Encodes Part Number and Serial Number. The latter is needed to get support for this Apple device.

GTIN-12/UPC-A:
Identifies the product at POS for checkout. You can identify this as a UPC because there are 12 digits shown.

Figure 1.9  Apple prefers to use Code 128 for supplementary information.

mechanism was added to reduce the distance that the laser would have to travel when scanning such a code, which could become rather long.

This code was adopted by GS1 to represent supplementary product information like product weights, dimensions, expiration dates, and so on. Multiple pieces of information like this can be contained in a single barcode. When used in this context, the concrete application of the Code 128 specification is referred to as GS1-128.

This standard defines multiple “application identifiers” to specify the meanings of values following them. The full list of identifiers is reproduced in appendix C. In human-readable captions, GS1-128 barcodes indicate application identifiers by enclosing them within parentheses. This is how you can easily identify a GS1-128 barcode.

CODE 25 AND ITF-14
Dr. David Allais invented Code 25 at Intermec in 1972, before the previously mentioned Code 39 and Code 93, also credited to him. This makes Code 25 the oldest barcode symbology supported by iOS. In 1998 Eastman Kodak Company patented the use of Code 25 for marking film canisters for automatic processing in film development machines.

The barcode can only encode pairs of digits. The first digit is encoded in five bars, and the second digit is encoded in five spaces interleaved with the bars. There are two widths for bars and spaces. The encoding table is constructed such that there will always be two wide spaces and bars for one pair of digits. It’s because of this that the code got its original name: Interleaved 2 of 5, sometimes abbreviated as ITF and more commonly referred to as Code 25 (see figure 1.10).
As one of the oldest barcode types in use today, Code 25 has a few flaws. Besides being limited to encoding only even numbers of digits, it doesn’t have marker bars to indicate to a reading device where the bars start and where they end. This can cause incomplete reads. Also, there’s no built-in error checking or correction.

Despite those flaws, Code 25 was chosen by GS1 to represent 14-digit GTINs. The predominant use is for marking shipping boxes that contain multiple identical products. Because there are always 14 digits to be represented, the even-numbers-only limitation is a non-issue. A check digit adds protection for incorrect scans. As an additional measure for battling incomplete scans, so-called bearer bars were added. Those are meant to abort the scan if the scan line doesn’t cross all bars, for example, if the scan angle is too steep.

For printing on labels, only the horizontal bearer bars (above and below) are required by the standard. For printing on corrugate boxes, vertical bearer bars (left and right) are also recommended to even out the printing plate pressure. Figure 1.11 shows an ITF-14 barcode with both vertical and horizontal bearer bars.

GS1 International refers to this scheme as ITF-14 and lists it as an important standard on equal footing with its other standards. It’s probably for this reason that Apple added support for both ITF-14 and “classic” Code 25 with iOS 8.

1.2.2 2D barcodes in iOS

A problem of barcodes encoding text strings of variable length is that there’s a maximum useful width for the markings, which is dictated by what they’re printed on. You can’t put a longer string onto an envelope than fits on the paper. The obvious solution for this, represented by PDF417, is to wrap the single scanning dimension onto multiple lines. Such a stacked linear code can be read by specialized (and thus more expensive) laser-based scanners.

When there’s no single line that can cross all parts of a barcode in one scan, the code is referred to as 2D, and as a rule of thumb you need a camera to read this digital content. Several barcode symbologies were designed to form two-dimensional squares, most notably QR and Aztec.

Apple added support for the PDF417, QR, and Aztec barcode types to iOS so it could represent modern digital boarding passes and tickets in Passbook.
PDF417

PDF417 was invented by Dr. Ynjum P. Wang at Symbol Technologies in 1991. This company was founded in 1975, a mere two years after the initial UPC was announced, with a primary goal of pursuing the blooming retail- and inventory-management market.

The name PDF417 is in no way related to Adobe’s Portable Document Format (PDF), which uses the same acronym. This similarity in name often confuses people first dealing with it. In this case, PDF417 is short for portable data file. The barcode consists of symbols that contain 4 bars and spaces each, with each symbol being 17 units long, hence the 417. An example is shown in figure 1.12.

The greatest advantage of PDF417 is that you can decide how wide and how high each individual line should be. And although it’s being patented, PDF417 is fully in the public domain and is thus free of all usage restrictions, licenses, and fees. Thanks to these features, PDF417 has become the 2D barcode of choice for a great variety of use cases.

PDF417 is one of the formats accepted by the United States Postal Service for printing postage. More than 200 airlines have settled on using it as the Bar Coded Boarding Pass standard since 2005. It’s also used by package services and on ID cards.

QR CODE

You’ve probably seen a QR Code before: a large square made up of small squares, with larger squares in three of its corners. See figure 1.13 for an example. Those squares are used by scanners to align themselves on the code and determine its orientation.

In contrast to PDF417, the QR Code never had roots in the one-dimensional space—it was designed from the ground up to be read by CCDs. Toyota’s subsidiary Denso Wave, an automotive components manufacturer, invented the QR Code in 1994 for tracking parts around car factories.

QR Codes can represent any kind of data with great density. As the length of encoded data grows, so does the square area of the code, or the individual squares inside the code shrink. This means that the data density is only limited by the resolution of the scanner camera. The individual squares making up a QR Code are also referred to as modules.

The QR Code gained popularity outside of the auto industry when it was used to encode website addresses, mostly in print media and advertising. For a long time it was derided as being a Japanese fad, but now almost all phones have the ability to read QR Codes.
Because QR Codes can represent any data, use cases range from encoding vCards to audio files. For example, a QR Code sticker on a box could contain an audio recording of verbal instructions for visually impaired people.

There’s no universal standard for how certain kinds of complex data should be represented in a QR Code—it’s agnostic to what data you want to put in it.

QR Codes embed multiple error-correction symbols, which make them extremely resilient. See chapter 5, section 5.1.2, for an explanation.

**AZTEC CODE**

The Aztec Code was invented in 1995 by Andrew Longacre, Jr., and Robert Hussey. It was published by Automated Industry Machines (AIM) two years later. Its original purpose inside this company is not publicly known.

You can differentiate Aztec Codes from QR Codes by looking at the number of concentric squares. Aztec has one square in the center (see figure 1.14), whereas QR Codes contain multiple squares (see figure 1.13).

The name of the code derives from the central square bull’s-eye being reminiscent of an Aztec pyramid. Of course, the Aztec Code has built-in error correction as well.

This code is also one of the three barcode types that can be used on boarding passes following the Bar Coded Boarding Pass standard. (The two others are QR Code and Data Matrix.) On top of that, the Aztec Code is used by a great number of railway companies for digital train tickets.

Just like with the QR Code, you can represent any kind of data with Aztec Codes, so it’s a matter of developer preference which of the two you use to represent your data.

**DATA MATRIX**

Dennis Priddy of International Data Matrix, Inc., invented the Data Matrix barcode in 1994 and submitted it as a public domain standard to AIM (Association for Automatic Identification and Mobility) to promote widespread use.

Data Matrix is a very compact code that retains a high scan rate even if printed very small or with low contrast between dark and light blocks. The US Electronic Industries Alliance (EIA) recommends using Data Matrix for labeling small electronic components.

Similar to QR Codes, there are data redundancy and error correction provisions built into the specification. Because of this, Data Matrix symbols with scratches, tears, holes, and stains can be successfully read without data loss, even if more than 20% of the symbol were to become damaged and unreadable.

Data Matrix can be read at lower contrast ratios than most barcode symbologies, which is a helpful feature for environments where symbols may be obscured by grease,
dirt, paint and chemical coatings, and when the symbology is applied to metal and other reflective surfaces (see figure 1.15).

GS1 International promotes the use of Data Matrix in labeling health care products. The same application identifiers that were defined for Code 128 (see appendix C) can also be used in Data Matrix Codes, but will take up less space due to compaction and the use of two dimensions.

Apple added support for scanning Data Matrix Codes in iOS 8.

### 1.2.3 So many choices: which barcode should I use?

Out of the large number of different barcode symbologies, Apple chose to support the ones that make most sense for mobile applications. If you’re building apps to read existing codes, it’s very likely that one of the barcodes we’ve already discussed has you covered:

- Physical products, all items that are scanned at the point of sale: GTIN family
- Relatively short alphanumeric texts, serial numbers: Code 93 or Code 128
- Digital tickets, boarding passes, loyalty cards, PassKit: PDF417, QR Code, or Aztec Code
- Arbitrary data in a square space: QR Code or Aztec Code
- Arbitrary data using less height than width: PDF417

iOS, beginning with version 7, can generate bitmaps of all the 2D barcodes we’ve discussed, and it provides scanning ability for all the 1D and 2D barcodes we’ve covered. To fill the niche of generating 1D barcodes, I created a library dubbed BarCodeKit. This commercial library is available to readers of this book at no charge. You’ll find all the details about this in chapter 5, which covers generating barcodes for display and print.

### 1.3 Summary

2014 marked the 40-year anniversary of the first widely used barcode, which eventually became the GTIN. Its original goal was to increase the efficiency of the grocery industry by enabling automatic product identification at the point of sale, and it fulfilled this goal many times over as the entire world adopted it. Most barcode symbologies are ISO standards at the lowest technical level; GS1 is in charge of defining the semantic meaning of content represented as GTINs and Code 128.

Other industries had different needs, and this led to the development of barcode symbologies that could represent alphanumerics characters. Code 39 is the oldest among the barcode types supported by iOS; Code 93 and Code 128 are more advanced symbologies.
The advancements in digital image processing gave rise to a new kind of barcode using more than one dimension. Small, inexpensive cameras on a chip, called CCDs, were able to scan 2D barcodes as well as the older 1D barcodes, which initially could only be scanned with a laser beam. As it became standard for smartphones to have a camera built-in, this put a potential barcode scanner in everyone’s pocket.

Before the rise of the mobile phone, barcodes were only useful in places that had scanning equipment installed. Point-of-sale (POS) systems had bulky cash registers with built-in laser scanners and a database for looking up price information. But these technologies are now available in modern smartphones. Not only can users now scan barcodes with a device they’re already carrying with them, but always-on internet connectivity and device sensors detect the user’s current context and add degrees of utility that have never been possible before.

These are the key takeaways for this chapter:

- Barcodes are a tried-and-true technology, with the most widely used form—the UPC—being more than 40 years old.
- Previously barcodes required laser-based scanners found in factories or at the point of sale. Today camera-equipped mobile phones are able to read them with ease. This opens up new usage scenarios with app users being able to interact with the physical world.
- One-dimensional (1D) barcodes encode numbers or alphanumeric characters on a single line. Two-dimensional (2D) barcodes are able to encode arbitrary data on a grid forming a square.
- The international GS1 organization oversees the semantic implementations of barcodes in the context of commerce. See appendixes B and C for details of the semantics that they manage. GS1 unified the previously used UPC, EAN, and JAN codes and numbering schemes into the GTIN.
- Apple began to integrate barcode technologies in iOS 7, adding only a few barcode-related APIs to iOS 8. Beginning with iOS 7, you don’t need any third-party software to add barcode scanning to your apps. This book will equip you with all you need to know to create barcode-enabled apps.
Barcodes with iOS
Oliver Drobnik

Barcodes are a universal way to track and share information, appearing on everything from cereal boxes to shop windows. Starting with iOS 7, Apple has added native features for building apps that scan, display, and print barcodes, eliminating the need for third-party libraries.

Barcodes with iOS teaches you how to effectively use barcodes in your iOS apps. You’ll master Apple’s new barcode frameworks while you explore real-world examples that integrate code scanning and generation and metadata retrieval into your apps. Along the way, you’ll pick up numerous best practices for bringing together the physical and digital worlds.

What’s Inside

• Learn about all barcode formats supported by iOS
• Native barcode scanning with AV Foundation
• Using Core Image and BarCodeKit to produce a wide range of barcodes
• Printing to sheets and labels with AirPrint
• Retrieving metadata for products with NSURLSession and NSURLProtocol
• Harnessing context information from Core Location and iBeacons

This book is written for readers with a working knowledge of Objective-C and iOS app development.

Oliver Drobnik is an independent consultant specializing in custom iOS and Mac development.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/BarcodeswithiOS

“Highly relevant and inspiring.”
—Jim Matlock, IBM

“All you need to know, in a single book.”
—Johan Pretorius
Next Step Design

“A great book for both beginning and seasoned developers.”
—Christopher W. H. Davis, Nike

“A must-have book.”
—Arif Shaikh
Sony Pictures Entertainment

“A unique resource with lots of valuable information. Code examples are clear and precise.”
—Gavin Whyte
Commonwealth Bank of Australia

Free eBook
SEE INSERT