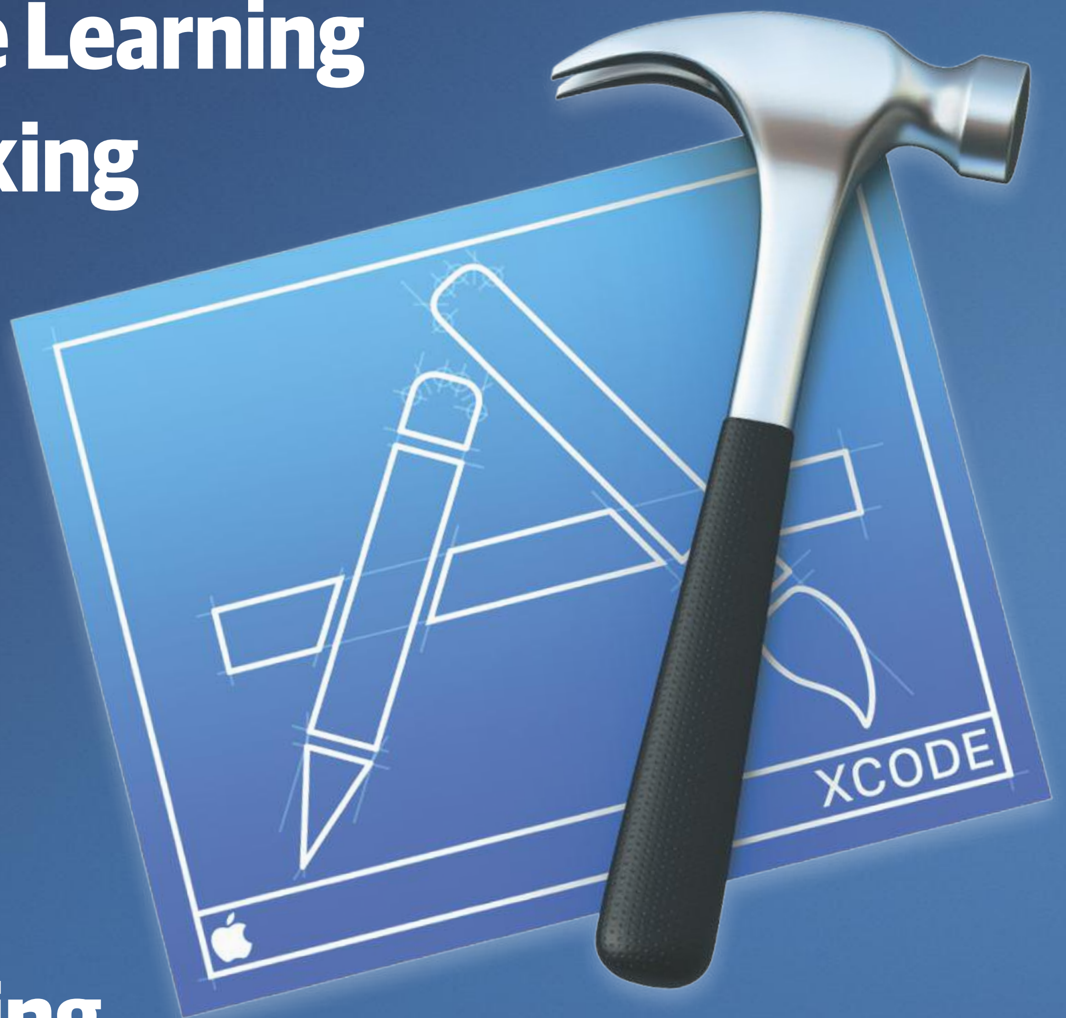
Mac & i kompakt

Das Apple-Magazin von c't

Software-Entwicklung

- CoreML
- Machine Learning
- Networking
- ARKit
- UI
- iPadOS
- Siri
- Swift
- NFC
- Debugging



IMMER EINE RUNDE GESCHiCHTE.

2× Mac & i mit 25 % Rabatt testen und Geschenk sichern!

Ihre Vorteile:

- **Plus:** digital und bequem per App
- **Plus:** Online-Zugriff auf das Artikel-Archiv*
- **Plus: Geschenk nach Wahl**, z.B. einen BestChoice-Gutschein im Wert von 10 € oder ein 5-in-1 Objektiv-Kit
- **Lieferung frei Haus**

Für nur 16,20 € statt 21,80 €

* Für die Laufzeit des Angebotes.

Jetzt bestellen und von den Vorteilen profitieren:
0541 80 009 120 · leserservice@heise.de

www.mac-and-i.de/miniabo

Mac & i
Das Apple-Magazin von c't.

Mac & i kompakt

Das Apple-Magazin von c't

Software-Entwicklung

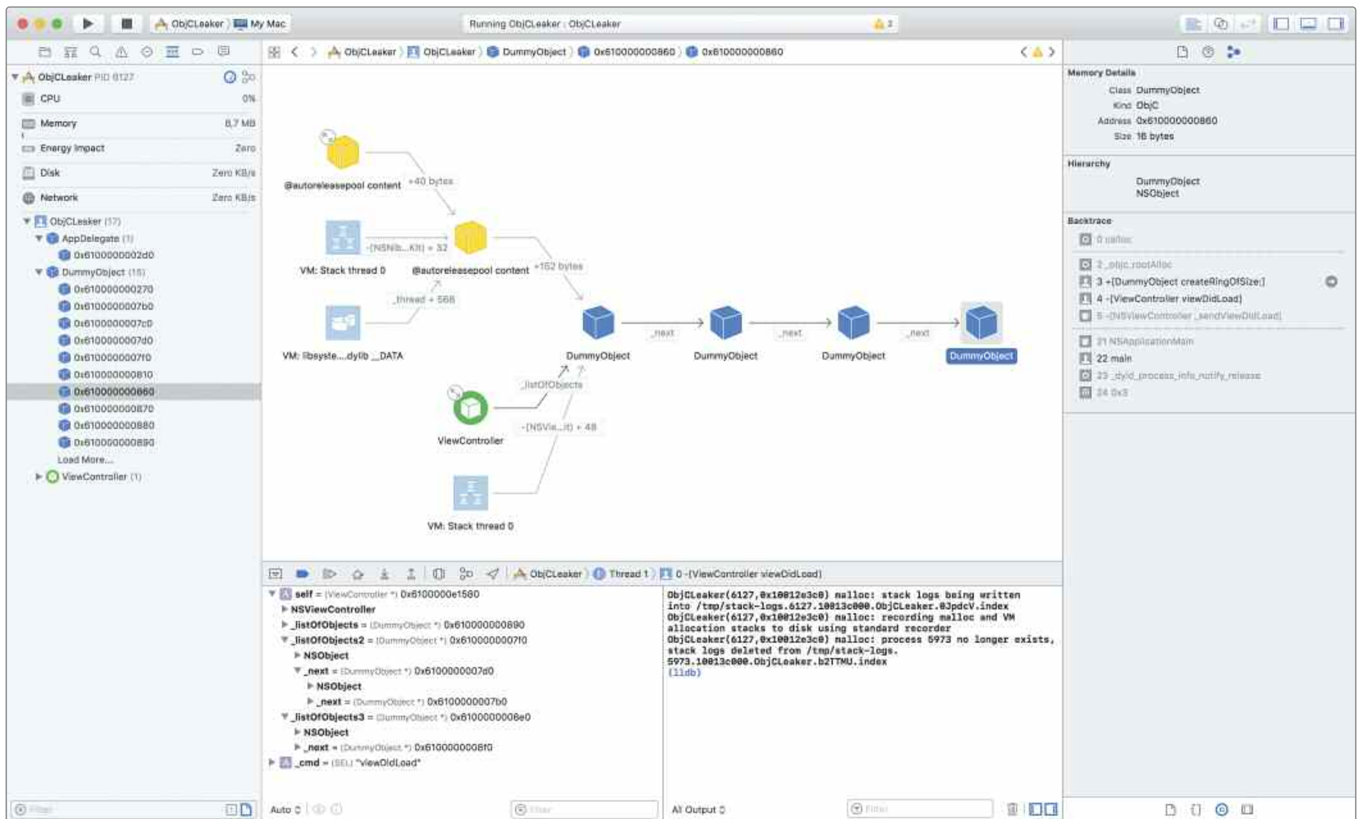
Liebe Leserin, lieber Leser,

die Rubrik Developer's Corner widmet sich in jeder Ausgabe von Mac & i einem speziellen Problem oder einem Framework von Apple – aus der Praxis von Entwicklern für Entwickler geschrieben. In diesem ePaper fassen wir alle Artikel seit Heft 1/2017 chronologisch sortiert zusammen. Wir haben sie in ihrem Ursprungszustand belassen. Manches dürfte in der Zwischenzeit ein wenig anders oder vielleicht von Apple ergänzt worden sein. Die Quellcode-Beispiele und Demo-Projekte, die Sie über den Webcode am Ende der Artikel herunterladen können, mögen nicht immer auf Anhieb funktionieren, der Migrationsassistent von Xcode sollte aber die meisten Probleme beheben.

Wir wünschen Ihnen viel Spaß bei der Lektüre,
Ihre Mac & i-Redaktion

RAM-Inventur – Speicherlecks mit dem Memory Graph Debugger finden .... S. 4
Bunte Mitteilungen – So nutzen Sie Rich Notifications in Apps .... S. 10
Wuuusch und weg – UI-Animationen selbst erstellen .... S. 16
Auf den Schirm! – Externe Displays mit der eigenen iOS-App ansteuern .... S. 22
Gut erkannt – Gesichts- und Objekterkennung mit dem Vision-Framework .... S. 28
Denk-Apparat – Machine Learning mit CoreML .... S. 36
Neue Welten – Das neue ARKit Framework in iOS 11 .... S. 42
iPad-Apps für Profis – Anwendungen an die neuen Funktionen von iOS 11 anpassen .... S. 50
Baukastensystem – Bedienoberflächen schneller gestalten mit ContainerViews .... S. 56
Apps mit Verständnis – Natural Language Processing in eigenen Projekten nutzen .... S. 62
Gehirnjogging für die App – Eigene Machine-Learning-Modelle mit CreateML erstellen .... S. 68
„Hey, Siri… das Übliche, bitte!“ – So machen Sie Ihre App fit für Siri und Kurzbefehle .... S. 74
Augmented Reality reloaded – Die Neuerungen von ARKit 2 in iOS 12 .... S. 80
Richtig netzwerken – Mobiles Internet in Apps – so geht's .... S. 86
Kreuz und quer – Apples MultipeerConnectivity-Framework .... S. 92
Swift 5 – Die Highlights im Praxiseinsatz .... S. 98
Von Watch bis 6K – Bedienoberflächen erstellen mit SwiftUI .... S. 104
Modern verschlüsseln – Sichere Swift-Apps mit CryptoKit .... S. 110
NFC olé – Apples erweiterte Nahfunk-Unterstützung in Apps nutzen .... S. 116
Besser debuggen – So integrieren Sie Debug-Menüs in Apps .... S. 122
Progressive Web Apps – Cross-Plattform-Programme über iOS und macOS hinaus .... S. 128

Die Artikel stammen aus den Mac & i-Heften 1/2017 bis 3/2020.



RAM-Inventur

So finden Sie Speicherlecks in Ihrer App

Mit Xcode 8 hat Apple ein neues Werkzeug unter die Entwickler gebracht: den Memory Graph Debugger. Er vereint nicht nur die Funktionen einiger Terminal-Kommandos, sondern hat auch eine schicke Bedienoberfläche innerhalb der Entwicklungsumgebung. Wir haben uns angeschaut, was er zu bieten hat und ob er die externe Instruments-App ersetzen kann.

Von Markus Stöbe

Wer Software entwickelt, macht Fehler. Viele davon sind relativ leicht mit Hilfe des in Xcode integrierten Debuggers zu finden, sobald sie auftreten (siehe Mac & i Heft 2/14, S. 152). Manch andere dagegen sind sehr schwer zu entdecken, beispielsweise Speicherlecks, auf Englisch memory leaks.

Solche Fehler treten immer dann auf, wenn eine Anwendung freien Speicher vom Betriebssystem anfordert und ihn anschließend niemals wieder zurückgibt. Problematisch wird das allerdings nicht sofort. Sowohl das OS als auch andere Apps arbeiten zunächst ungehindert weiter, solange noch genügend freier Speicher vorhanden ist. Verwendet man die problematische App jedoch länger, frisst diese mehr und mehr RAM. In der Folge kann das gesamte System träger reagieren. Im schlimmsten Fall können Programme abstürzen oder vom Betriebssystem aufgrund von Speichermangel ohne Rückfrage beendet werden.

Blöd ist daran vor allem, dass nicht zwingend die App abstürzt, die das Problem verursacht hat, sondern die, die keinen Speicher mehr bekommt. Extrem speicherhungrige Apps wie Spiele empfehlen deshalb mitunter, das Gerät neu zu starten, bevor man sie startet. Vor allem unter iOS hört man diesen Rat immer wieder. Der Neu-

start reinigt den Speicher von allen Leichen und ist damit das Einzige, was Anwender gegen solche Fehler unternehmen können.

Für Entwickler sind Speicherlecks wegen der verzögerten Auswirkungen sehr schwer zu entdecken. Apple arbeitet deshalb schon seit vielen Jahren auf allen seinen Plattformen daran, ihnen an dieser Stelle unter die Arme zu greifen und das Problem zu entschärfen. Mehr über Speichermanagement an sich und über die bisherigen Ansätze, es zu verbessern, finden Sie im Kasten auf Seite 154.

Wie ein Leck entsteht

So sehr Apple sich auch bemüht, Automatismen werden Datenlecks nie ganz vermeiden können. Ein einfaches Beispiel macht deutlich, warum das so ist: Nehmen Sie an, Ihre App würde eine verkettete Liste implementieren. Die dazugehörige Klasse hat lediglich zwei Properties: ein Datenfeld für hypothetische Nutzdaten sowie einen Zeiger auf den folgenden Datensatz in der Liste. In Swift sähe das in etwa so aus:

```
class DummyObject: NSObject {
    public var next: DummyObject?
    public var data: Data?
  ...
}
```

Erzeugt man nun eine Reihe von Instanzen diesen Typs und trägt dabei immer die Zeiger auf die folgenden Objekte korrekt ein, hat man zunächst kein Problem. Jede Instanz wird von einer anderen referenziert und kann deshalb nicht automatisch gelöscht werden. Besonders ist lediglich der erste Eintrag, denn der hat keinen Vorgänger. Er wird nur durch die App referenziert. Sobald diese Referenz aufgelöst wird, setzt eine Kettenreaktion ein, die alle Instanzen der Reihe nach löscht.

Trägt man zuvor aber in den letzten Datensatz den ersten als Nachfolger ein, hat man einen Kreis von Referenzen gebildet, der sich nicht mehr automatisch auflösen lässt. Selbst wenn nun die App ihre Referenz löscht, bleibt der Kreis erhalten, denn jedes Objekt wird schließlich noch von einem anderen referenziert. Entsprechend haben alle Referenz-Zähler einen Stand größer 1. Für das Betriebssystem bedeutet das „Achtung, dieses Objekt wird noch benutzt", ergo bleibt es im Speicher. Andernfalls würde die App und am Ende auch der Anwender einfach so seine Daten verlieren.

Dieses Szenario bildet unsere Beispiel-App für macOS nach, deren Quellcode Sie über den Webcode am Ende des Artikels laden können. Sie finden dort je eine Version in Swift und Objective-C. In ihr baut eine Schleife eine vorgegebene Anzahl von Objekten und verkettet diese zu einem Kreis. Eine Referenz auf das erste Objekt liefert die Methode zur weiteren Verwendung an den ViewController zurück, der diese in einer Variablen speichert. Über den einzigen Button der Anwendung („Leak") kann man diese Referenz auf nil setzen. So können Sie bequem vor und nach dem Auftreten des Speicherlecks einen Blick in die Ausgabe des Memory Debuggers werfen.

Analyse starten

Um die Suche nach Speicherlecks zu starten, benötigen Sie mindestens Xcode 8.2. Eingeführt wurde der Memory Graph Debugger zwar schon in Version 8.0, allerdings mit zahlreichen Fehlern, die ihn vor allem mit Swift-Code relativ unbrauchbar machten. Grundsätzlich können Sie dann sofort mit einem beliebigen Programm loslegen. Mit den richtigen Einstellungen im Projekt leistet der Debugger sogar noch mehr, dazu gleich weitere Infos.

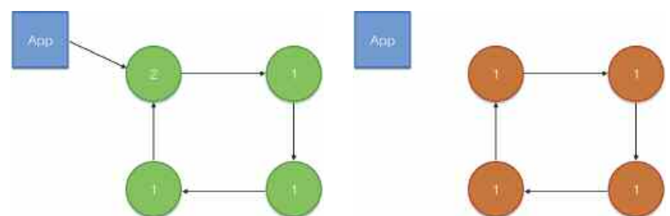
kurz & knapp

- Speicherlecks entstehen durch schwer zu findende Fehler in Programmen.
- Oft sind zirkulare Referenzen der Grund, warum Speicher nicht freigeräumt werden kann.
- Mit dem Memory Graph Debugger unterstützt Apple den Entwickler bei der Fehlersuche direkt in Xcode.
- Er zeigt grafisch, wie Objekte und Referenzen miteinander verknüpft sind, und kennzeichnet verwaiste Referenzen. Das erleichtert das Finden des problematischen Codes.

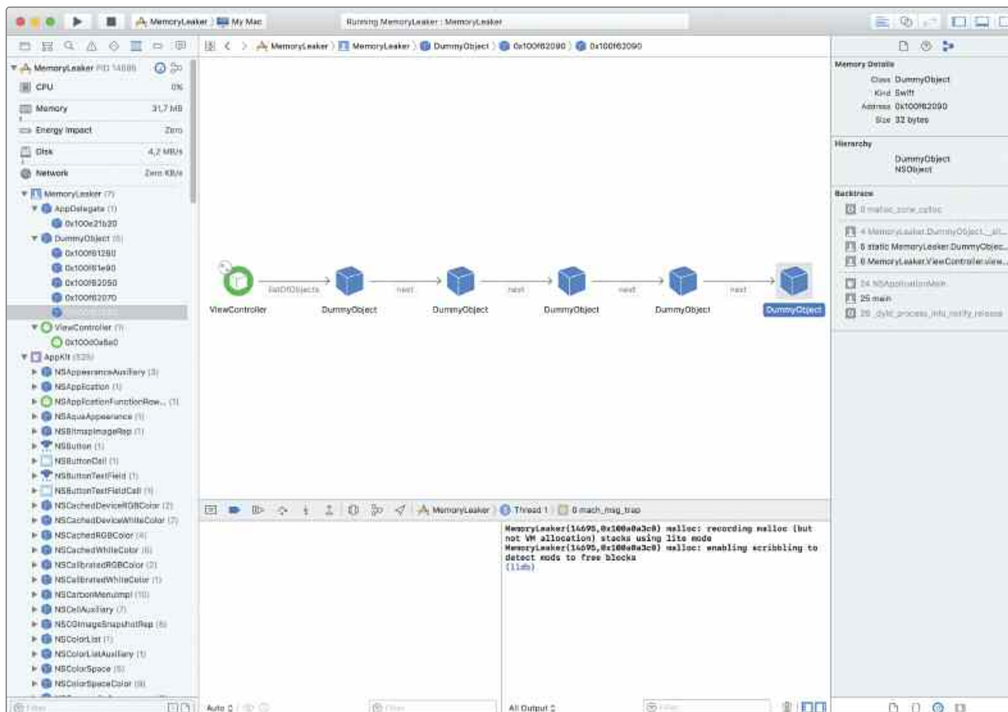
Die Funktionsweise ist einfach: Sie können sich zu einem beliebigen Zeitpunkt zur Laufzeit des Programmes den Speicher-Inhalt mit einem Klick auf den „Debug Memory Graph"-Knopf anzeigen lassen – das ist der zweite von rechts in der Knopfleiste unter dem Quellcode-Bereich. Dadurch hält Xcode die Ausführung des Programms vorübergehend an. Wieder anlaufen lassen können Sie es – genau wie nach einem Breakpoint – über die Playtaste fünf Knöpfe weiter links. Apropos Breakpoint: Sie können Ihr Programm freilich auch durch solch einen Haltepunkt stoppen und erst dann den Memory Graph anzeigen lassen. Der Vorteil dieser Variante besteht darin, dass Sie mehr Informationen über die im Speicher befindlichen Daten bekommen als ohne den Breakpoint.

Statt des Quellcodes blendet Xcode zum Debugging eine grafische Darstellung der Objekte und ihrer Beziehungen zueinander ein. Icons repräsentieren die Objekte, Pfeile zwischen diesen stehen für die Referenzen der Objekte. An einigen Stellen fasst die Entwicklungsumgebung den Graphen zusammen, um die Übersichtlichkeit zu verbessern. Erkennbar ist das an einem Doppelpfeil in einem Kreis, der links oben am Objekt-Icon angebracht ist. Ein Klick darauf blendet versteckte Teile des Graphen ein. Ein weiterer Klick darauf faltet ihn wieder zusammen.

Ein Doppelklick auf das Icon eines der Objekte setzt dieses in den Mittelpunkt. Xcode zeichnet den Speichergraphen dann ausgehend von diesem Objekt neu, was mitunter automatisch zuvor ausgeblendete Bereiche sichtbar macht. Wer also gesuchte Objekte im Graphen vermisst, kann zunächst durch einen Doppelklick auf ein anderes Objekt weitere Details finden.



Verweisen Objekte einer App (blau) reihum aufeinander (grün), bleibt der Referenz-Kreis auch dann bestehen, wenn die App sie nicht mehr referenziert. Die Referenzzähler der jeweiligen Objekte sind schließlich alle größer als null, also kann das Betriebssystem sie nicht entfernen – obwohl sie für die App nutzlos sind (rot).



Hält man die Ausführung einer App über den Knopf des Memory-Graph-Debuggers an (zweiter Knopf von rechts in der Leiste unter dem Graphen), stellt Xcode die aktuell im Speicher befindlichen Objekte als Graph dar. In der Liste in der Spalte ganz links kann man die Ansicht auf ein anderes Objekt umschalten.

Hilft das nicht, kommt man über die Liste links neben dem Speichergraphen weiter. Sie listet alle aktuell im Speicher auffindbaren Objekte der Anwendung. Wählt man eines davon an, zeigt Xcode im Editorbereich rechts daneben den dazu passenden Ausschnitt des Speichergraphen an. Wie auch bei Fehlerlisten kann man diese Liste über ein Textfeld filtern, etwa indem man dort den Namen der gesuchten Klasse eingibt. Für die Beispiel-App wäre eine Filterung nach „DummyObject" hilfreich. Am rechten Rand des Textfelds befinden sich noch zwei weitere Buttons, über die sich die Liste bequem filtern lässt. Der mit dem Ausrufezeichen zeigt nur Probleme in der Speicherverwaltung an und blendet alles andere aus. Der Button, der aussieht wie ein leeres Dokument, beschränkt die Liste auf Einträge, die durch den Quellcode des Programms verursacht wurden – vom Betriebssystem

Speichermanagement im Detail

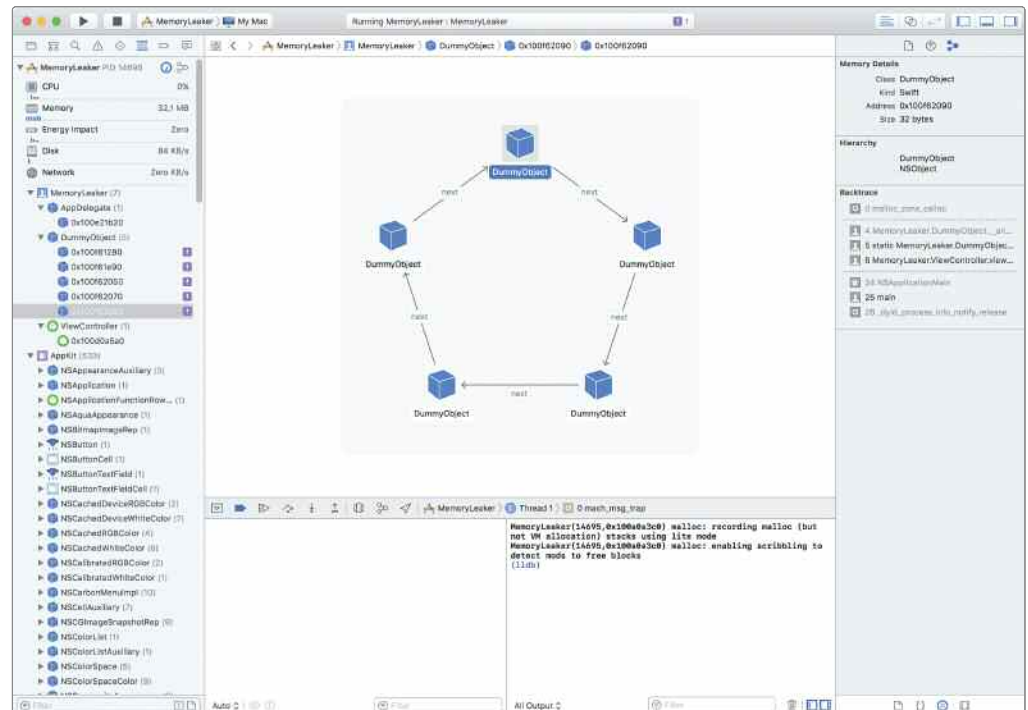
Programmierer haben schon relativ lange mit dem Speichermanagement zu kämpfen. Vor der Einführung der objektorientierten Programmierung mussten sie mittels malloc von Hand Speicherbereiche beim Betriebssystem anfordern, etwa wenn eine Anwendung Daten über das Netzwerk nachladen wollte. Sobald sie den Speicher nicht länger benötigte, musste der Entwickler ihn via free-Funktion beim Betriebssystem freigeben. Tat er das nicht, blieb der Speicherbereich bis zum nächsten Neustart des Rechners belegt und war weder für Apps noch das OS sichtbar. Er war also de facto verschwunden, daher der Name Speicherleck: RAM war wie Wasser durch ein Loch in einem Eimer entwischt.

Speicherlecks selber finden

Zu diesen Zeiten kam man dem Problem relativ leicht auf die Spur: Programm starten, Download abwarten, Progamm beenden und vorher wie nachher den freien Speicherplatz prüfen. Waren die Zahlen unterschiedlich, hatte man ein Leck. Da es im Programm nur relativ wenige malloc/free-Befehle gab, kam man dem Problem schnell auf die Schliche.

Moderne objektorientierte Programmiersprachen arbeiten wesentlich dynamischer. Für jedes Objekt, das der Entwickler instanziert, wird automatisch Speicher beim OS angefordert. Ebenso automatisch soll der Speicher wieder freigegeben werden, sobald er nicht mehr benötigt wird. Das klingt in der Theorie problemlos, in der Praxis ist es aber ein heikles Spiel. Der Computer selbst, genauer gesagt der Compiler oder die ausführende Runtime-Umgebung, ist in diesem Fall dafür zuständig, zu erkennen, wann der Zeitpunkt gekommen ist, ein Objekt zu löschen.

Das ist alles andere als trivial, denn die Maschine weiß ja nichts über die Pläne des App-Entwicklers. Gut möglich, dass er schon beim Start der App ein Objekt anlegt, welches er erst ganz am Ende beim Beenden der App wieder benutzt. Verschärfend kommt hinzu, dass Anwendungen in modernen Betriebssystemen ja gar nicht mehr immer sofort beendet werden. Sie können im Hintergrund laufen oder schlafen und zu späteren Zeitpunkten erneut geweckt werden. Kurzum: Der Rechner braucht Hilfe. Im Laufe der Zeit gab es dafür zahlreiche Ansätze, wie man die Speicherverwaltung verbessern kann.



Wenn Sie in der Liste links eines der problematischen Objekte auswählen (lila Ausrufezeichen), zeigt Xcode den Referenz-Zyklus an. Zusätzliche Informationen listet der Object-Inspector am rechten Rand. Welche Methode das Objekt erzeugt hat, erfährt der Entwickler aber nur mit den richtigen Projekt-Einstellungen.

beziehungsweise den Frameworks angeforderten Speicher oder angelegte Objekte blendet er aus.

Probleme erkennen

Wenn Sie den Memory Graph Debugger einblenden, bevor das Speicherleck entsteht, weist nichts auf ein Problem hin. Die Liste der Dummy-Objekte wird nicht einmal als Zyklus dargestellt. Man kann erkennen, dass der ViewController über die Variable listOfObjects eine Referenz auf die erste Instanz einer Reihe von Dummy-Objekten hält. Jedes dieser Objekte hält wiederum in der Variablen next einen Zeiger auf die folgende Instanz und so weiter.

Drücken Sie vor dem Start des Debuggers jedoch den Leak-Knopf der Beispiel-Anwendung, sieht das Bild im Debugger ganz

Halb-Automatik

Zunächst mussten Entwickler sogenannte Retain-Counter pflegen: einen Zähler in jedem Objekt, der signalisiert, ob noch jemand das Objekt verwendet. Wer das Objekt noch benötigt, zählt den Zähler einmalig hoch (retain), wer fertig ist mit der Nutzung, dekrementiert den Zähler um eins (release). Sinkt der Zähler auf null, gibt das Betriebssystem den Speicher frei und löscht das Objekt.

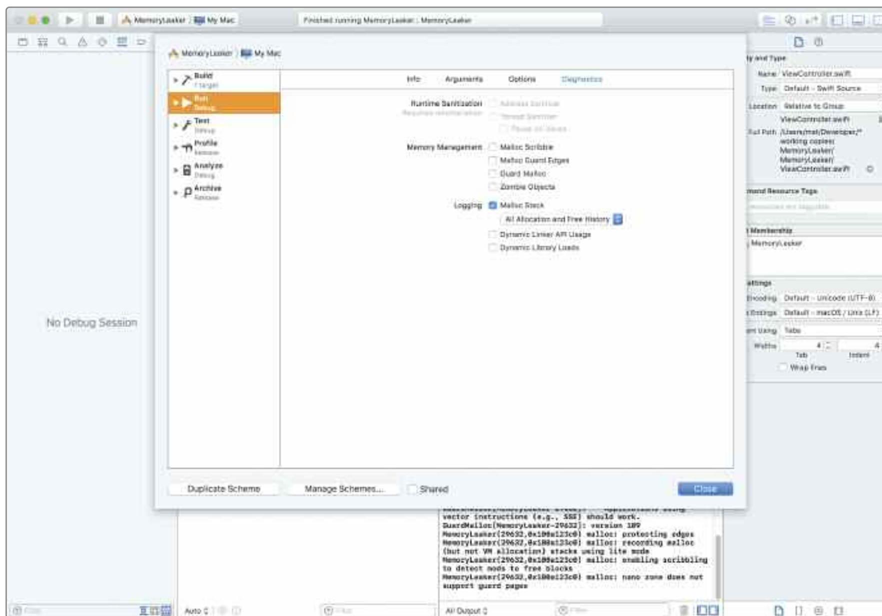
Dieses Verfahren birgt mehrere Schwächen. Zum einen ist es nicht immer klar, wer wann den Zähler erhöhen und erniedrigen muss, etwa bei Methoden, die eine neue Objekt-Instanz erzeugen und zurückgeben wie -dataWithContentsOfURL:. In diesem Fall muss die erzeugende Methode den Zähler erhöhen, sonst übergibt sie ein Objekt mit einem Retaincount von 0, was wiederum zur sofortigen Löschung durch das OS führen würde. Trotzdem sorgt der sogenannte Autorelease-Pool dafür, dass so erzeugte Objekte mit einer Zeitverzögerung automatisch freigegeben werden. Der Empfänger muss also trotzdem für das Objekt die Retain-Funktion aufrufen und es zu gegebener Zeit freigeben.

Zum anderen kann es sehr leicht passieren, dass Objekte sich reihum gegenseitig im Speicher halten – ein sogenannter Retain-Zyklus. Der Kreis von Referenzen als solcher ist nicht schädlich. Problematisch wird er nur dann, wenn die App keinen Zugriff mehr auf ihn hat und ihn deshalb nicht mehr auflösen kann.

Voll-Automatik

Automatic Reference Counting, kurz ARC, wurde Mitte 2011 eingeführt. Damals waren gerade iOS 5 und OS X Lion neu erschienen. Es erlöst den Programmierer weitestgehend von der Last, sich selbst um die Speicherverwaltung kümmern zu müssen. Das verwirrende Spiel mit Retain, Release und Autorelease war damit vorbei. Referenz-Zyklen kann man aber auch damit erzeugen und somit auch Speicherlecks. Auf diese müssen Entwickler also nach wie vor ein Auge haben und genau dabei hilft der neue Speicher-Debugger in Xcode.

Wer im Detail erfahren möchte, wie ARC und die übrigen Methoden der Speicherverwaltung funktionieren, sollte einen Blick in Mac & i Heft 5/2012 ab Seite 158 werfen.



Aktiviert man das Mitschreiben der Speicheranforderungen im Scheme der auszuführenden App, kann der Memory Graph Debugger sogar die Methode anzeigen, die ein herrenloses Objekt erzeugt hat.

anders aus. Dann markiert Xcode die unerreichbaren Instanzen in der linken Spalte im Debug Navigator durch ein lila unterlegtes Ausrufezeichen. Wählt man eine der markierten Instanzen an, zeichnet Xcode nun den Zyklus gut sichtbar ins Fenster. Man erkennt auch sofort, dass dieser Referenz-Zyklus keinerlei Verbindung zur App mehr hat.

Die eigentlich wichtigen Fragen lauten an dieser Stelle aber: Woher kommt das Objekt eigentlich und warum existiert es immer noch? Im konstruierten Beispiel hat die App den Fehler mit der zirkularen Referenz selbst herbeigeführt. In echten Apps fällt die Analyse deutlich schwerer – hier kann der Debugger helfen.

Wählt man ein Objekt links im Debug Navigator oder im Graphen an, zeigt der „Memory Inspector“ am rechten Rand von Xcode weitere Informationen an. Er listet zusätzliche Informationen zur Instanz, von oben nach unten sind das etwa deren Klasse, die verwendete Sprache, die Speicheradresse des Objekts und dessen Größe in Bytes. Darunter findet der Entwickler den Bereich „Hierarchy“, in dem die Erbfolge der Instanz klar wird. Die Klasse DummyObject aus dem Beispiel-Projekt leitet sich direkt von NSObject ab, deshalb enthält die Liste in diesem Fall nur zwei Zeilen.

Diese Daten können bereits einen ersten Aufschluss darüber geben, wo der Fehler zu suchen ist. Meist gibt es ja nur wenige Stellen, an denen das herrenlose Objekt erzeugt worden sein könnte. Ab dieser Stelle kann man dann den Lebenszyklus der Instanz auf herkömmlichem Wegen nachverfolgen, etwa mit Breakpoints.

Übrigens lassen sich auch die Bezeichner an den Pfeilen zwischen zwei Icons im Graphen anklicken, obwohl sie wie inaktiver Text aussehen mögen. Für diese listet der Inspektor, von wo nach wo die Referenz verläuft und ob sie „strong“ oder „weak“ ist. Verlaufen mehrere Referenzen entlang einer Kante, vermerkt Xcode die Anzahl im Label. Klickt man auf die Zahl, klappt eine Liste der Referenzen auf, die man einzeln begutachten darf.

Per Ctrl- oder Rechtsklick auf eines der Icons im Graphen bietet der Debugger weitere Optionen an, etwa zur Definition der Klasse zu springen. Manche Objekte kann man so auch via Quicklook anschauen, etwa `UIImage`-Instanzen.

Spurensicherung

Noch nützlicher wäre, man könnte gleich direkt zu der Stelle im Quellcode springen, an der die Instanz erzeugt wurde. Um diese Funktion zu aktivieren, muss man zunächst etwas umständlich eine Einstellung im Scheme ändern.

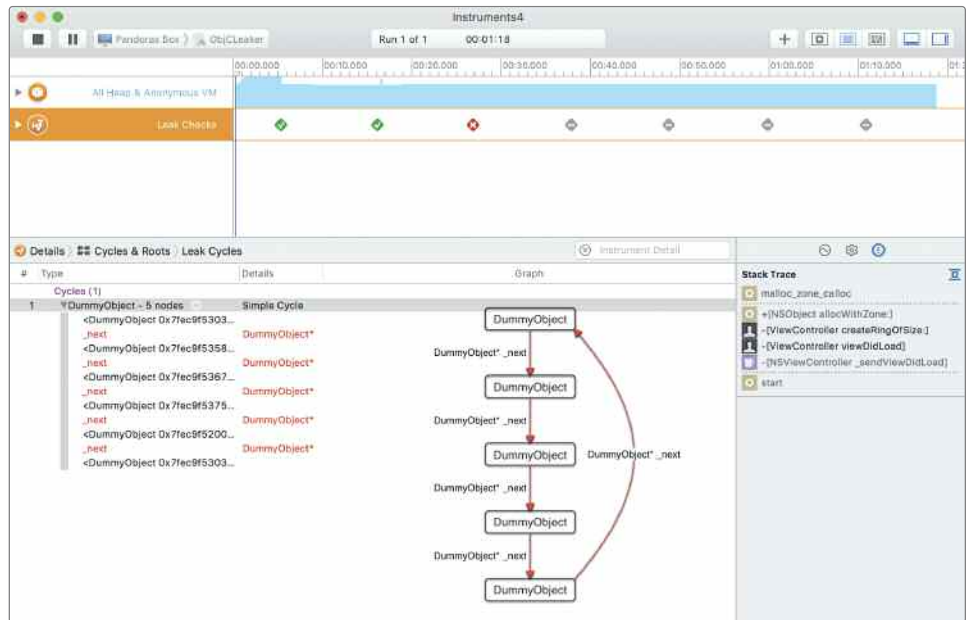
Dazu klicken Sie links oben auf den Button rechts neben der Stop-Taste, welche den Namen Ihrer App trägt. Im aufspringenden Menü wählen Sie „Edit Scheme …“ und gelangen so in den Scheme-Editor. Dort wählen Sie in der linken Liste die Run-Sektion, die daraufhin rechts neben der Liste sichtbar wird. Im Tab „Diagnostics“ aktivieren Sie in dessen Logging-Bereich schließlich „Malloc Stack“. Welchen Wert Sie im daraufhin aktivierten Dropdown-Menü auswählen, ist letztlich egal. Die Optionen stellen ein, dass im Log nur noch lebende Objekte enthalten sein sollen oder auch solche, die schon wieder freigegeben wurden. Der Unterschied ist aber letztlich nur relevant, wenn man das Log selbst auswerten will. Für die Arbeit mit dem Memory Graph Debugger ist das unwichtig.

Schließen Sie das Fenster über den Close-Knopf unten rechts. Mit diesen Einstellungen speichert Xcode während der Ausführung der App den „Backtrace“ auf der Festplatte, also eine Liste der Methoden, die in irgendeiner Weise Speicher angefordert haben.

Schaut man sich die Informationen zu einem Objekt im Memory Graph Debugger mit diesen Einstellungen an, erscheinen zusätzliche Informationen im Memory Inspector. In der untersten Sektion („Backtrace“) listet die Entwicklungsumgebung nun eine ganze Reihe von Methodenaufrufen, die zur Erzeugung des Objekts geführt haben. Solche, die das Betriebssystem im Hintergrund ausgeführt hat, sind ausgegraut. Die Methoden aus dem eigenen Quellcode sind in schwarzem Text gehalten. Schiebt man den Mauszeiger über eine solche Methode, erscheint ein Button am rechten Ende der Zeile, mit dem man zu der entsprechenden Stelle im Quellcode springt. Der Sprung führt genauer zu der Stelle, an der das Objekt erzeugt wurde. Nicht etwa zu der, die das Speicherleck zur Folge hatte. Was letztlich zum Speicherleck geführt hat, muss der Entwickler also nach wie vor selbst herausfinden. Das liegt vor allem daran, dass Xcode beziehungsweise der Memory Graph Debugger nicht aufzeichnet, wann genau beziehungsweise in welcher Methode und an welcher Codezeile das Leck entsteht.

Wunschliste

Bisher mussten Entwickler für die Suche nach Speicherlecks den Debugger Instruments verwenden. Der bietet in diesem Bereich in etwa die gleichen Funktionen wie der nun integrierte Memory Graph Debugger. Eine Funktion aber fehlt dem Neuling: Er sucht nicht selbstständig im Hintergrund nach Speicherlecks. In Instru-



Der bisherige Debugger Instruments ist nicht ganz so bequem zu starten wie der neue Memory Graph Debugger. Dafür kann er automatisch nach Speicherlecks suchen, während man die App benutzt.

ments voreingestellt sind 10-Sekunden-Intervalle. Der Entwickler kann währenddessen seine App ganz normal benutzen und wird automatisch auf die Probleme aufmerksam gemacht.

Apple listet Memory Leaks im Issue Navigator von Xcode zwar als „Runtime"-Fehler, also als Fehler, die zur Laufzeit der App aufgetreten sind. Aktualisiert wird die Liste der gefundenen Speicherlecks aber nur, wenn man die Ausführung der App durch den Klick auf den Debugger-Button einfriert. Um problematische Situationen dennoch schnell zu finden, prüft man am besten Positionen, an denen die App viele Objekte erzeugt oder löscht. Das könnte beispielsweise nach dem Herunterladen von neuen News-Meldungen oder Dokumenten sinnvoll sein.

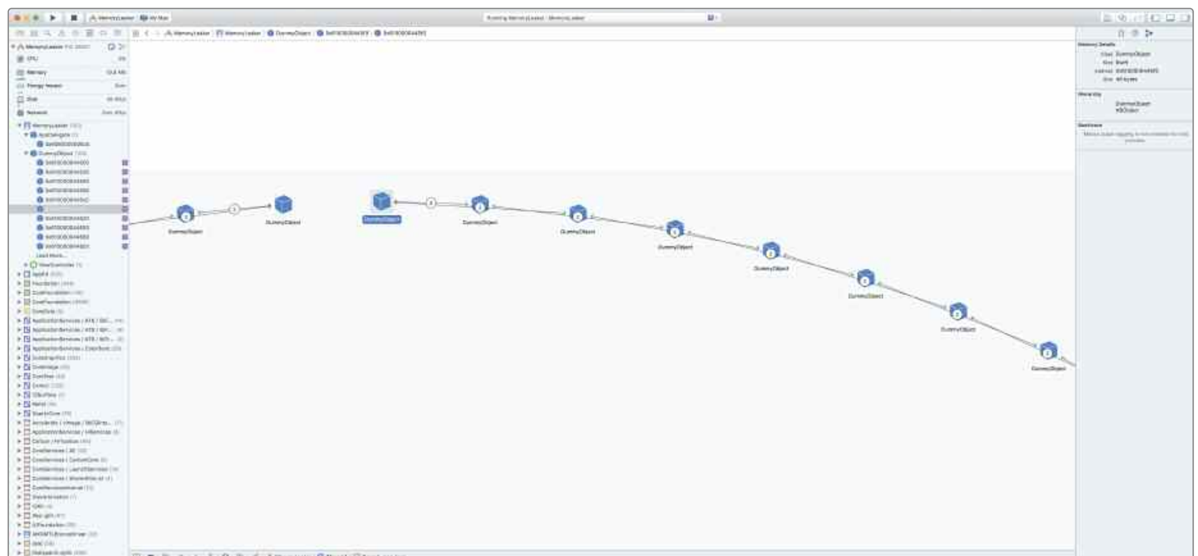
Zusätzlich hat der Memory Graph Debugger noch ein kosmetisches Problem: Der Anwender kann die Größe des Graphen nicht ändern. Eine Zoom-Funktion hat Apple schlicht nicht vorgesehen. Wer das Pech hat, einen recht großen Graphen generiert zu bekommen, blickt mitunter auf eine (scheinbar) leere Anzeige. In Wahrheit ist Xcode mit dem Zeichnen längst fertig und erwartet, dass man zu einer bestimmten Stelle scrollt.

Zudem sollte man sich nicht zu stark an der Form des Graphen orientieren. Einen Referenzzyklus malt der Debugger – wie oben erwähnt – erst dann als Kreis, wenn es zu einem Leck kommt. Bis dahin sieht der Entwickler nur eine Kette von Objekten. In einigen Tests klappte Xcode den zyklischen Graphen in der Mitte zusammen, um Platz zu sparen. Dadurch geriet auch die Reihenfolge der Icons auf den ersten Blick völlig durcheinander und ich vermutete zunächst einen Flüchtigkeitsfehler meinerseits bei der Verknüpfung der Objekt-Instanzen untereinander.

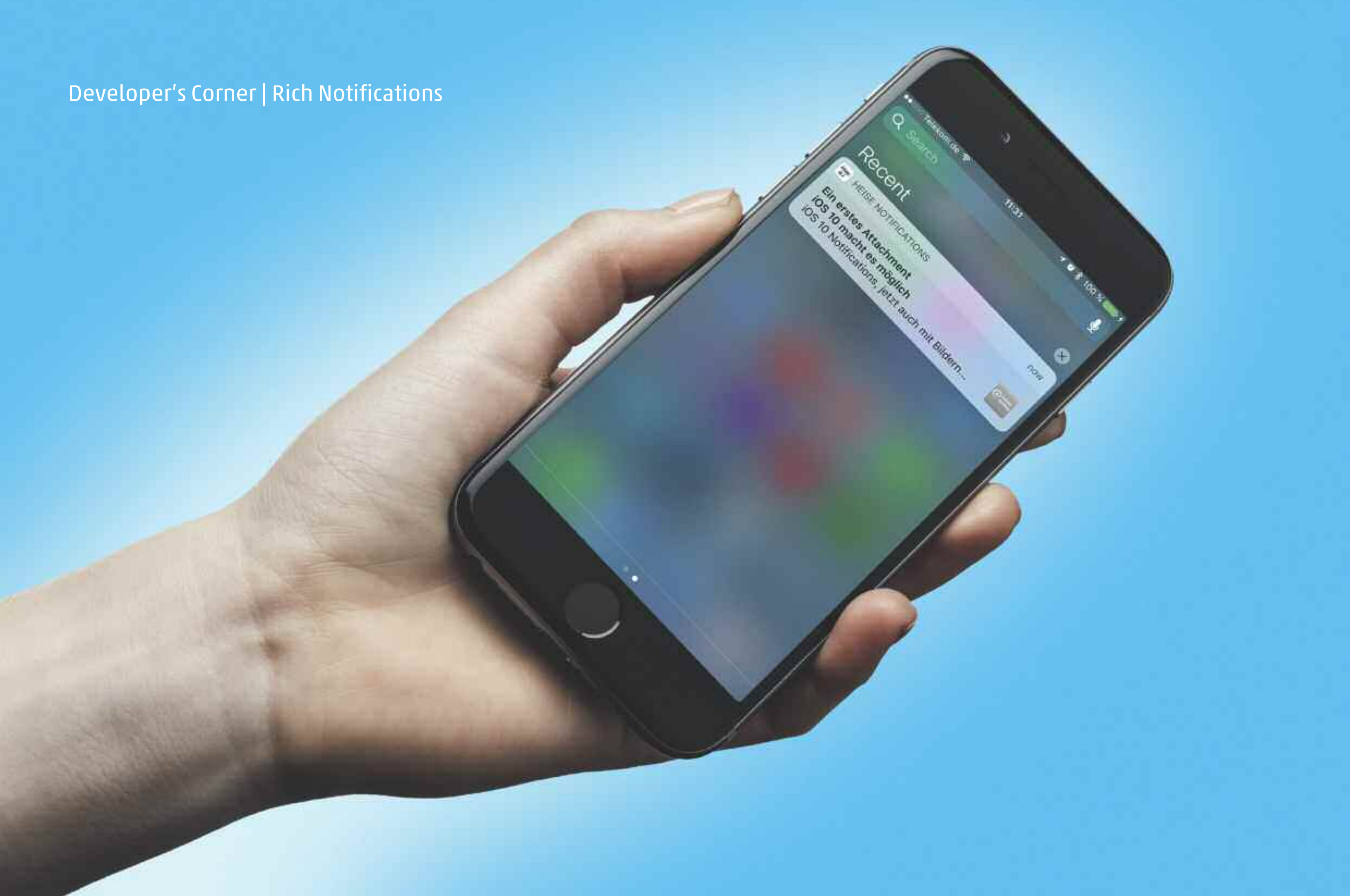
Unterm Strich

In der aktuellen Fassung kann sich der Memory Graph Debugger recht gut mit Instruments messen. Im Alltag erweist sich die Integration in Xcode als nützlich, etwa wenn man beim Testen seiner App aus heiterem Himmel wachsenden Speicherbedarf beobachtet. In den meisten Fällen weiß man dann nicht mehr genau, wie man das Problem erzeugt hat. Ein Klick startet den Debugger, ohne dass man die App neu starten müsste, wie es bislang mit Instruments der Fall war. So erhalten Entwickler relativ leicht ausreichende Hinweise, um das Speicherproblem zu lösen.

Um auf alle Probleme und nicht nur die auffälligsten aufmerksam zu werden, greift man aktuell besser noch zu Instruments, denn das kann selbstständig und in regelmäßigen Abständen nach Speicherlecks suchen. (thk)



Von sehr großen Graphen bekommt man mitunter gar nichts zu sehen, denn eine Zoom-Funktion hat Apple nicht implementiert. Da hilft nur, die Ränder manuell abzusuchen.



Bunte Mitteilungen

So nutzen Sie Rich Notifications in Apps

Mitteilungen sind heutzutage nahezu in jeder App zentraler Bestandteil. Bislang war man bei der Darstellung von Notifications auf Text limitiert – seit iOS 10 sind Bilder, Audio, Videos und sogar Custom Views möglich. Wir zeigen, wie viel das neue Notifications-Framework für iOS 10 bietet und was es zu beachten gilt.

Von Gero Gerber

In iOS 10 hat Apple für Mitteilungen das UserNotifications-Framework eingeführt. Dieses ersetzt die existierende Funktionalität, die bislang im UIKit Framework angesiedelt war. Auch das neue Framework unterscheidet grundsätzlich zwischen Local und Remote Notifications (auch als Push Notifications bezeichnet). Local Notifications initiiert eine App auf dem Gerät selbst, etwa um den Benutzer an einen Termin zu erinnern. Remote Notifications dagegen gelangen über Apples Push Notifications Service (APNs) an das Device, zum Beispiel aktuelle News, Börsenkurse oder Sport-Updates. Solche Mitteilungen können für den Anwender sichtbar oder unsichtbar sein. Silent Pushes diesen dazu, einer App im Hintergrund mitzuteilen, dass Neuigkeiten auf dem Server bereit stehen.

Grundlagen und Vorarbeiten

Damit eine App überhaupt Notifications initiieren und empfangen kann, muss sie sich zunächst beim Betriebssystem registrieren.

Hierbei kann sie eine Kombination aus mehreren Optionen zur Interaktion mit dem Benutzer anfragen: Banner erlauben das visuelle Darstellen einer Mitteilung. Badges stellen Zahlen direkt am App-Icon dar. Sounds und Vibrations ermöglichen eine akustische beziehungsweise haptische Rückmeldung. Die Option CarPlay unterstützt die Darstellung im Auto.

```
// Eine Referenz auf UNUserNotificationCenter holen
let notificationCenter = UNUserNotificationCenter.current()

// Notification Optionen erbitten
notificationCenter.requestAuthorization(options: [.alert, .badge, .sound])
{
    // Der Completion Handler gibt Rückmeldung, ob der Benutzer zugestimmt hat
    // (granted == true) oder nicht (granted == false)
    (granted, error) in
    if granted == false
    {
        print("Failed to authorize: \(error)")
    }
}
```

UNUserNotificationCenter stellt also die zentrale Klasse zur Verwaltung aller Notification-spezifischen Aufgaben dar. Im Zuge der ersten Autorisierungsanfrage einer App muss der Anwender seine Erlaubnis erteilen. Diese und weitere Details lassen sich auch nachträglich in den Mitteilungs-Einstellungen ändern. Eine App kann diese Optionen auslesen, um darauf intelligenter zu reagieren.

Damit iOS über Ereignisse informiert, nutzt man den Delegate-Mechanismus, eine Form des Beobachter-Entwurfmusters. Remote Notifications benötigen weitere Voraussetzungen: Zunächst muss man im Apple Developer Center eine neue App-ID erzeugen. Dann gilt es, die Option „Push Notifications“ zu aktivieren – zusätzlich auch im Xcode-Projekt im „Capabilities“-Tab. Weiterhin benötigt man ein SSL-Zertifikat, um über eine gesicherte Verbindung mit APNs zu kommunizieren (weitere Details dazu siehe Kasten auf Seite 153).

Die Registrierung einer App für Remote Notifications erfolgt mithilfe der UIApplication-Klasse. Hierfür muss eine Internet-Verbindung zum APNs bestehen, da dieser der App bei erfolgreicher Registrierung ein Device Token zukommen lässt. Grundsätzlich gilt: Remote Notifications lassen sich nicht im iOS Simulator, sondern nur auf einem Device testen.

```
UIApplication.shared.registerForRemoteNotifications()
```

War die Registrierung für Remote Notifications erfolgreich, erhält man über die Methode application(_:didRegisterForRemoteNotificationsWithDeviceToken:) ein Device Token vom APNs. Schlägt die Registrierung fehl, kommt die Methode application(_ application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: Error) zur Ausführung. Beide Methoden sind Bestandteil des UIApplicationDelegate-Protokolls, welches von der AppDelegate-Klasse implementiert wird.

Beim Device Token handelt es sich um eine eindeutige Kennung, damit APNs die Remote Notifications an ein bestimmtes Device senden kann – aus Sicherheitsgründen verwendet Apple hier aber nicht die UDID des Gerätes. Das Token wird an ein App-Backend (etwa Microsoft Azure oder Amazon Web Services) gesendet, damit dieses aktiv Remote Notifications an das Device schicken kann – unabhängig von der App. Die Kennung bleibt so lange gleich, bis der Anwender die App löscht und eventuell neu installiert. Um Remote Notifications leichter testen zu können, hilft das Pusher-Projekt für macOS (siehe Kasten auf Seite 155).

kurz & knapp

- Apple hat mit dem User Notifications Framework ein komplett neues Framework für iOS 10 eingeführt
- Notifications können nicht nur Text, sondern auch Bilder, Audio und Videos darstellen
- Mitteilungen lassen sich auch anzeigen, während die App im Vordergrund läuft
- Notification Extensions ermöglichen die Kontrolle über den Inhalt und die Darstellung von Mitteilungen
- Entwickler können das User Interface von Notifications anpassen

Die erste Notification

Eine simple Local Notification besteht aus drei Elementen: dem eigentlichen Inhalt (Notification Content), dem Auslösemechanismus (Trigger) und einer Anfrage für die Mitteilung (Notification Request).

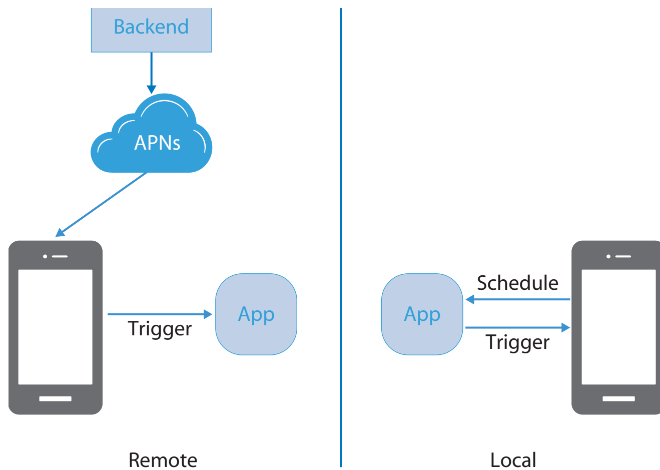
```
// 1. Content erzeugen
let content = UNMutableNotificationContent()
content.title = "Notifications"
content.subtitle = "Was ist neu?"
content.body = "Was man alles mit Notifications machen kann..."
content.badge = 1;
// 2. Trigger definieren
let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
// 3. Request anlegen, basierend auf dem Content und Trigger
let requestIdentifier = "meinRequest"
let request = UNNotificationRequest(identifier: requestIdentifier, content: content,
    trigger: trigger)
// Notification Request einplanen
UNUserNotificationCenter.current().add(request)
{
    // Wurde die Notification erfolgreich angelegt?
    (error) in
    print(error ?? "Ok")
}
```

Zusätzlich muss bei einer Remote Notification der Inhalt in einer Payload definiert sein:

```
{
    "aps" : {
        "alert" : {
            "title" : "Notifications",
            "subtitle" : "Was ist neu?",
            "body" : "Was man alles mit Notifications machen kann..."
        },
        "badge" : 1,
        "sound" : "default"
    },
}
```

Wird entweder die Local Notification ausgelöst oder eine Remote Notification empfangen, zeigt das Betriebssystem diese dem Benutzer als Overlay an. Hierbei gibt es drei Zustände zu unterscheiden:

– Das Device ist entsperrt und die App im Hintergrund: es erscheint für eine kurze Zeit ein Notification-Overlay am oberen Bildschirmrand.



Remote Notifications werden durch den Apple-Push-Benachrichtigungsdienst (APNs) initiiert, Local Notifications durch iOS.

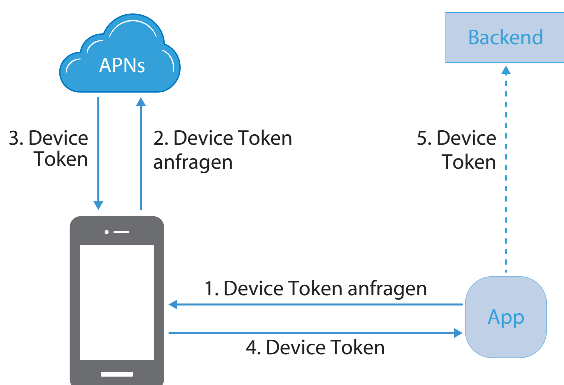
– Das Device ist gesperrt: Die Notification wird im Lock Screen angezeigt.
– Der Benutzer hat die Notification verpasst und öffnet die Mitteilungszentrale: Alle Mitteilungen sind dort chronologisch sortiert gelistet.

Mit iOS 10 lassen sich Notifications auch anzeigen, wenn die App im Vordergrund läuft. Hierfür müssen Sie die Methode userNotificationCenter(_:willPresent:withCompletionHandler:) des UNUserNotificationCenterDelegate-Protokolls implementieren.

Anhand des Typs notification.request.trigger lässt sich feststellen, ob es sich bei der Notification um eine Local oder Remote Notification (UNPushNotificationTrigger) handelt.

Notification Trigger

Wie schon erwähnt, muss man beim Anlegen eines Notification Requests (UNNotificationRequest) einen Trigger-Parameter mit übergeben. Dieser entscheidet, wann die Mitteilung letztendlich auslöst und zur Anzeige kommt. Mit iOS 10 existieren vier verschiedene Trigger-Typen: Remote Notifications kennen nur den Push-Trigger, lokale Hinweise zusätzlich den Time-Interval-, Calendar- und Location-Trigger. Der UNTimeIntervalNotificationTrigger dient dazu, eine Notification nach Ablauf eines Zeit-Intervalls auszulösen. Ein UNCalendarNotificationTrigger zeigt eine Notification zu einem bestimmten Datum inklusive Uhrzeit in der Zukunft an. Last, but not least werden anhand von UNLocationNotificationTrigger Notifications genau dann ausgelöst, wenn der Benutzer einen definierten Ort betritt oder verlässt. Auf Wunsch aktiviert sich dieser Geo-Trigger anschließend erneut. Alle anderen Arten zünden nur einmalig.



Das Device Token für Remote Notifications erhält man vom APNs.

Mehr Action

Neben der reinen Anzeige lassen sich Notifications ab iOS 10 mit Buttons versehen, mit denen der Nutzer auf eine Notification reagieren kann. Jeder Knopf kann eine von drei Aktionen auslösen: Die Default Action öffnet nach einem Tap auf die Notification die zugehörige App. Eine Dismiss Action verwirft und entfernt eine Mitteilung. Mit Custom Actions (eingeführt in iOS 8) kann der Benutzer auf eine Mitteilung reagieren, ohne die App öffnen zu müssen. Seit iOS 9 kann eine Custom Action auch Text zurückgeben, zum Beispiel eine Schnellantwort auf eine iMessage-Nachricht. Aus Platzgründen lassen sich für eine Notification maximal vier Aktions-Buttons darstellen.

Um Notifications mit Actions zu versehen, müssen Sie diese zunächst registrieren und einer Kategorie zuordnen (siehe Methode initializeNotification() im Beispielprojekt).

Für jede Action legen Sie ein Objekt vom Typ UNNotificationAction an. Diesem Objekt wird im Initialisierer ein Identifier und ein Title übergeben. Der Identifier dient später dazu, den Action-Typ erkennen zu können. Der Title entspricht der Anzeige auf dem Button. Für Text Input Actions verwenden Sie die Klasse UNTextInputNotificationAction.

Actions weist man Kategorien zu, um empfangene Notifications zuordnen zu können. So erkennt das System, welche Actions es für eine bestimmte Kategorie anzeigt und welche Extensions es für Notifications einer bestimmten Kategorie anwenden soll.

Darüber hinaus lassen sich Actions mit dem Attribut .foreground versehen. Vordergrund-Actions führen dazu, dass beim Tap auf den Action-Button die App lädt und der Benutzer mit ihr interagieren kann. Als weitere Option steht der Wert .destructive bereit, um dem Benutzer klar zu machen, dass die Action beispielsweise Daten löscht, iOS hebt sie rot hervor. Die Option .authenticationRequired fordert den Benutzer dazu auf, das Device zu entsperren – erst dann wird die App über den Action Event informiert. Dies kann man etwa dazu nutzen, die App auf Daten zugreifen zu lassen, die im gesperrten Zustand verschlüsselt sind.

Soll das System eine Notification mit entsprechenden Actions anzeigen, so muss man bei Local Notifications den entsprechenden Identifier für die Kategorie angeben.

```
content.categoryIdentifier = "aktionsKategorie"
```

Für Remote Notifications muss entsprechend in der Payload der Schlüssel „category" auf die gewünschte Kategorie verweisen.

```
{
  "aps" :
  {
    "alert" : "Meine Notification",
    "category" : "aktionsKategorie"
  }
}
```

Auf Actions reagieren

Damit Actions eine Reaktion auslösen, muss man dies im Code vorsehen. Dazu implementiert man die Methode userNotificationCenter(_:didReceive:withCompletionHandler:) des Protokolls UNUserNotificationCenterDelegate. Sie übergibt den Parameter response vom Typ UNNotificationResponse.

Er enthält neben der ursprünglichen Notification die Property actionIdentifier, also den Identifier der vom Benutzer selektierten Action. Der response-Parameter bekommt den Typ UNTextInputNotificationResponse, wenn der Benutzer eine Text-Input-Action gewählt hat. Dabei enthält das Property userText die Texteingaben des Anwenders.

Am Ende rufen Sie den completionHandler auf, um dem Betriebssystem zu signalisieren, dass die Behandlung der Action abgeschlossen wurde.

Um eine Remote Notification per Silent Push zu übermitteln – damit eine App Updates im Hintergrund durchführen kann – muss die Payload den Schlüssel „content-available":1 enthalten. Darüber hinaus aktivieren Sie in Xcode im „Capabilities"-Tab die Einstellung „Background Modes" und den Modus „Remote notifications". Die Zeit, die eine App erhält, um eine Remote Notification im Hintergrund abzuarbeiten, limitiert Apple auf maximal 30 Sekunden.

Service Extensions

App Extensions ermöglichen es Entwicklern seit iOS 8, dem Betriebssystem und Apps neue Fähigkeiten beizubringen – etwa Foto-Tricks, Finder-Erweiterungen oder eine spezielle, virtuelle Tastatur.

In iOS 10 kommt die „Service Extension" hinzu. Sie modifiziert den Content einer Remote Notification, bevor der Benutzer sie sieht. So lassen sich etwa Fotos oder Videos in die Mitteilung einbetten, aber genauso wäre eine spezielle End-to-End Encryption machbar. Um iOS zu signalisieren, dass es eine Service Extension benötigt, enthält die Payload der Remote Notification den Schlüssel „mutable-content":1.

Um eine Service Extension einem App Projekt hinzuzufügen, wählen Sie in Xcode unter „File/New/Target…" im Tab „iOS" die „Notification Service Extension" aus. Legen Sie einen „Product Name" und für unser Beispiel die Programmiersprache Swift fest.

Im Ergebnis landet die neue Datei „NotificationService.swift" im Projekt. Sie enthält die Definition der Klasse NotificationService, die wiederum von UNNotificationServiceExtension abgeleitet ist. Der Einstiegspunkt in diese Klasse stellt die Methode didReceive(_:withContentHandler:) dar. Sie wird aufgerufen, sobald eine entsprechende Remote Notification eingeht. Der Parameter request enthält die empfangene Notification und der Callback Parameter contentHandler dient dazu, iOS mitzuteilen, dass die Notification bereit zur Darstellung ist, der Content also von der Extension abschließend angepasst wurde.

Für den Fall, dass die Modifizierung des Contents zu lange dauert, etwa weil die App Daten herunterladen muss, ruft iOS aus einem anderen Thread heraus die Methode serviceExtensionTimeWillExpire() der Klasse NotificationService auf. Dies ist quasi die letzte Chance, den Content zu modifizieren und gegebenenfalls auf eine Fallback-Lösung zurückzugreifen, die in der Ausführung nicht lange dauern darf. Auch diese Methode muss am Ende den contentHandler aufrufen, um iOS mitzuteilen, dass die Bearbeitung abgeschlossen ist.

Dauert auch diese Methode zu lang, zeigt iOS die ursprüngliche Notification an. Im folgenden Beispiel zeigen wir eine Notification mit einem sogenannten Media Attachment (einem Bild oder Video) an. Dazu definieren Sie eine Remote-Notification in JSON-Schreibweise mit der folgenden Payload und versenden diese:

```
{
    "aps":
    {
        "alert":
        {
            "title": "Ein erstes Attachment",
            "subtitle": "iOS 10 macht es möglich",
            "body": "iOS 10 Notifications, jetzt auch mit Bildern..."
        },
        "mutable-content":1
    },
    "my-
image":"http://www.heise.de/icons/ho/heise_online_facebook_social_graph.png"
}
```

Außerhalb des „aps"-Blocks findet sich der Schlüssel „my-image". Dieser enthält die URL, die auf eine PNG Datei verweist.

Um hier der Einfachheit halber eine unsichere http-Adresse verwenden zu können, legen Sie in der Datei „Info.plist" des Targets „NotificationService" das Dictionary „App Transport Security Settings" mit dem Schlüssel „Allow Arbitrary Loads" und dem Wert

APNs und Zertifikate für Remote Notifications erstellen

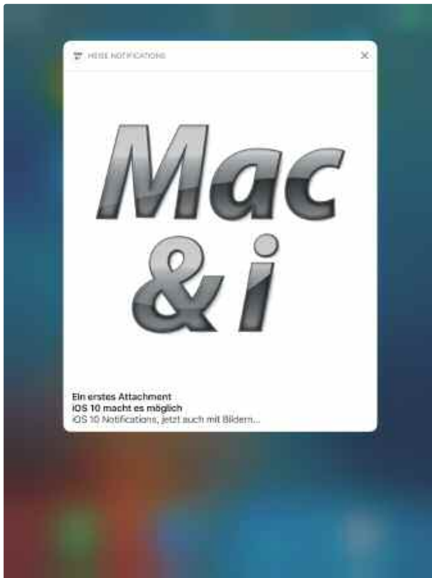
Um Remote Notifications über eine verschlüsselte Verbindung an APNs zu senden, braucht es Zertifikate. Das Development SSL Certificate benötigt man für den Entwicklungsprozess, das Production SSL Certificate für den Live-Betrieb der App. Apple bietet zwei APNs-Server an: Der Development Server ist unter gateway.sandbox.push.apple.com:2195 und der Production Server unter gateway.push.apple.com:2195 erreichbar.

Um eine Remote Notification zu initiieren, müssen Sie eine gesicherte Verbindung zu einem der beiden APNs-Server mit Hilfe des jeweiligen Zertifikats aufbauen. Erst dann lässt sich die Payload oder auch Nutzlast, also der eigentliche Inhalt der Remote Notification, im JSON-Format versenden.

Um ein Zertifikat anzulegen, öffnen Sie die Apple-Developer-Website und loggen sich mit Ihrem Developer Account ein. Anschließend öffnen Sie den Link „Certificates, Identifiers & Profiles". Nun selektieren Sie unter „Identifiers" den Link „AppIDs". Dort wählen Sie die gewünschte App ID und klicken auf „Edit". Nun muss unter dem Punkt „Push Notifications" ein Zertifikat für „Development" erzeugt werden. Folgen Sie den Anweisungen auf der Website. Am Ende dieses Prozesses landet das erzeugte Zertifikat in Ihrer Keychain. Von dort aus exportieren Sie das Zertifikat wieder, um Remote Notifications beispielsweise mit der App „Pusher" (siehe Kasten Seite 155) vom Desktop aus zu testen.

Media-Attachment-Arten

Attachment	Dateityp	Maximale Größe
Audio	kUTTypeAudioInterchangeFileFormat	5 MByte
	kUTTypeWaveformAudio	
	kUTTypeMP3	
	kUTTypeMPEG4Audio	
Bild	kUTTypeJPEG	10 MByte
	kUTTypeGIF	
	kUTTypePNG	
Video	kUTTypeMPEG	50 MByte
	kUTTypeMPEG2Video	
	kUTTypeMPEG4	
	kUTTypeAVIMovie	



Mit Hilfe von Service Extensions lassen sich Bild-Inhalte in einer Notification anzeigen.

Über die Custom Actions kann die App den Zustand der Bedienoberfläche ändern. Tippt der Anwender auf „Gefällt mir", erscheint ein kleiner Stern am Mac & i-Logo.

„YES" an. Dieses Setting sollten Sie aber möglichst nur für Testzwecke verwenden.

Bild hinzufügen

Sobald die Service Extension die Payload empfängt, kann diese die referenzierte PNG-Datei herunterladen und als UNNotificationAttachment der Notification hinzufügen:

```
override func didReceive(_ request: UNNotificationRequest, withContentHandler
        contentHandler: @escaping (UNNotificationContent) -> Void)
{
    let url = URL(string: request.content.userInfo["my-image"] as! String)!
    self.contentHandler = contentHandler
    self.bestAttemptContent = (request.content.mutableCopy() as?
    UNMutableNotificationContent)
    URLSession.shared.downloadTask(with: url)
    {
        (location, response, error) in do
        {
            let mediaAttachment = try UNNotificationAttachment(identifier: "media",
            url: location!, options: [UNNotificationAttachmentOptionsTypeHintKey:
            kUTTypePNG])
            if let bestAttemptContent = self.bestAttemptContent
            {
                // Modify the notification content here...
                bestAttemptContent.attachments = [mediaAttachment]
                contentHandler(bestAttemptContent)  }
        }
    }
}
```

Im String url weist man den Wert des Schlüssels my-image zu, also die URL der PNG-Datei. Anschließend lädt man das Bild über die Methode downloadTask(with:) der Klasse URLSession herunter. iOS speichert die Datei in einem temporären Ordner, dabei ändert sich allerdings die Dateiendung von „.png" in „.tmp". In der Objektreferenz mediaAttachement landet das Objekt vom Typ UNNotificationAttachment.

Der Code initialisiert dieses Objekt mit einem Identifier, der im späteren Verlauf den erneuten Zugriff auf dieses konkrete Attachment ermöglicht. Dabei wird die URL auf die temporäre Datei und ein Hinweis auf den Datei-Typ mit übergeben ([UNNotificationAttachmentOptionsTypeHintKey:kUTTypePNG]). Abschließend wird das neu erzeugte Attachment dem Property attachments der Klasse UNMutableNotificationContent als Array zugewiesen. Das UNMutableNotificationContent-Objekt erhält man aus dem Parameter request und dessen Property content. Im letzten Schritt übergibt man das modifizierte UNMutableNotificationContent-Objekt als Parameter an contentHandler und signalisiert iOS damit, dass die Notification bereit zur Darstellung ist.
Nach einer kleinen Änderung lassen sich auch Animated GIFs, Audio oder Videos darstellen. Hierbei muss neben der URL in der Payload lediglich der Hinweis auf den Datei-Typ gesetzt sein.

Content Extensions

Neben der Service Extension existiert die sogenannte Content Extension. Diese ermöglicht es, eine eigene Bedienoberfläche darzustellen, mit der der Benutzer interagiert – allerdings stark eingeschränkt. Das UI verarbeitet per se keine Input Events, stattdessen lässt sich über Notification Actions der Zustand des UI verändern.

Eine Content Extension erzeugen Sie in Xcode über ein neues Target vom Typ Notification Content Extension. Dieses Target enthält neben einem Storyboard auch eine Klasse vom Typ UIViewController, die das Protokoll UNNotificationContentExtension implementiert. Damit iOS eine Remote Notification auch an die korrekte Content Extension mit dem entsprechenden UI weiterleitet, müssen Sie in der Datei „Info.plist" der Content Extension unter „NSExtension/NSExtensionAttributes/UNNotificationExtensionCategory" ein oder mehrere Strings aufführen, die auf eine Notification-Kategorie verweisen. Anhand der Kategorie in der Notification wird die entsprechende Content Extension ausgewählt und gestartet. Das UNNotificationContentExtension-Protokoll enthält zwei für uns interessante Methoden. didReceive(_:) kommt zur Ausführung, sobald eine Notification für die Extension eintrifft.