Programowanie urządzeń mobilnych (9)

Debugowanie kodu natywnego Java

W poprzedniej części kursu omówiliśmy sposób debugowania aplikacji tworzonych w Cordovie. Pokazaliśmy jak analizować kod JavaScript, HTML i CSS oraz ruch sieciowy, a więc wszystko to, co przeciętny twórca aplikacji Cordovy buduje samodzielnie. Zademonstrowane dotąd narzędzia umożliwiają wgląd w aktualnie wykonywany kod aplikacji i podgląd zmiennych, ale niestety nie rozwiązują wszystkich problemów, z jakimi borykają się programiści korzystający z Cordovy. Nierzadko okazuje się, że problem z błędnym działaniem aplikacji leży w bibliotekach i pluginach Cordovy, stworzonych w natywnym języku danej platformy sprzętowo-programowej. W artykule opisujemy, jak rozwiązywać tego typu trudności poprzez debugowanie Javy – języka natywnego dla platformy Android.

Jak i w poprzednich częściach kursu, koncentrujemy się na Androidzie, przy czym w tym przypadku cała treść artykuły praktycznie nie będzie miała jakiegokolwiek zastosowania do innych systemów operacyjnych. Można by się też zastanowić, jaki sens ma opisywanie debugowania w języku Java, którego czytelnicy mogą wcale nie znać, bo przecież kurs dotyczy Cordovy, a więc skupia się na zupełnie odmiennym JavaScripcie. W praktyce jednak umiejętność rozpoznania problemu, czy choćby wskazania jego źródła jest bardzo cenna. Narzędzia do debugowania aplikacji androidowych nie są proste w użyciu, ale można za ich pomocą wiele osiągnąć nawet bez znajomości języka Java. Pozwalają monitorować stan urządzenia mobilnego znacznie „głębiej" niż narzędzia opisane w poprzedniej części kursu, wykrywać procesy zużywające zbyt dużo zasobów, a przede wszystkim określać, które pluginy, w którym momencie powodują problemy. Szczególnie to ostatnie ma duże znaczenie dla osób piszących programy Cordovy, gdyż w przypadku niepoprawnej pracy urządzenia, pozwala dosyć dokładnie ocenić, w jakich warunkach zachodzi problem i zdecydować o ewentualnej zamianie używanej wtyczki Cordovy na inną. Oczywiście narzędzia do debugowania stanowią też niezbędną pomoc dla wszystkich, którzy chcą samodzielnie tworzyć pluginy Cordovy, ale o tym napiszemy w innej części kursu.

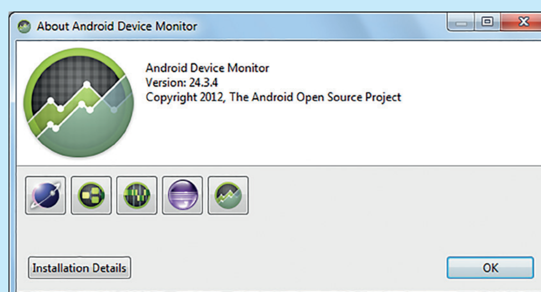
Android Device Monitor

Podstawowym narzędziem, jakie będziemy używali w niniejszej części kursu jest Android Device Monitor (**rysunek 1**) – oprogramowanie instalowane wraz z Android Studio. Jest to środowisko wykonane w oparciu o platformę Eclipse, dzięki czemu jest dosyć przenośne. Co więcej, moduły używane w Android Device Monitorze mogą być wykorzystywane bezpośrednio w Eclipse, co będzie pomocne dla tych osób, które programują z użyciem tego IDE (Zintegrowanego Środowiska Deweloperskiego). My jednak skoncentrujemy się na samodzielnej aplikacji w wersji 24.3.4, która została zainstalowana wraz z innymi programami opisanymi dotąd w trakcie kursu.

Program startuje się poleceniem **monitor**, wpisanym np. w polu „uruchom" systemu Windows. Powoduje ono uruchomienie pliku **monitor.bat**, znajdującego się w katalogu z narzędziami deweloperskimi Androida. Domyślnie jest to ścieżka *c:\Users\NazwaUżytkownika\AppData\Local\Android\sdk\tools\monitor.bat.* Plik **monitor.bat** sprawdza dostępność bibliotek Javy i uruchamia program **monitor.exe**, znajdujący się w podkatalogu zależnym od aktualnej architektury systemu – np. **lib\monitor-x86_64\monitor.exe** dla 64-bitowego systemu Windows.

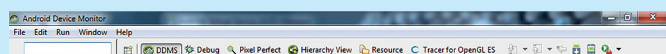
Android Device Monitor (ADM) składa się z jednego okna, obsługującego kilka perspektyw. Te znane są z Eclipse i określają sposób podziału głównego okna programu na moduły, realizujące poszczególne funkcje. Podstawowe perspektywy ADM to DDMS, Debug, Hierarchy View, Pixel Perfect, Resource i Tracer for OpenGL ES. Opiszemy w skrócie większość z nich, przy czym dla osób piszących aplikacje w Cordovie, najważniejsza będzie DDMS.

Zanim to zrobimy, warto jeszcze wspomnieć o ogólnej budowie okna ADM. Główne menu nie obejmuje wielu pozycji. Pozwala jedynie na wprowadzenie ustawień środowiska, uruchomienie zewnętrznych programów lub właśnie wybór i konfiguracje perspektyw. Lista ustawień środowiska jest bardzo długa i pozwala

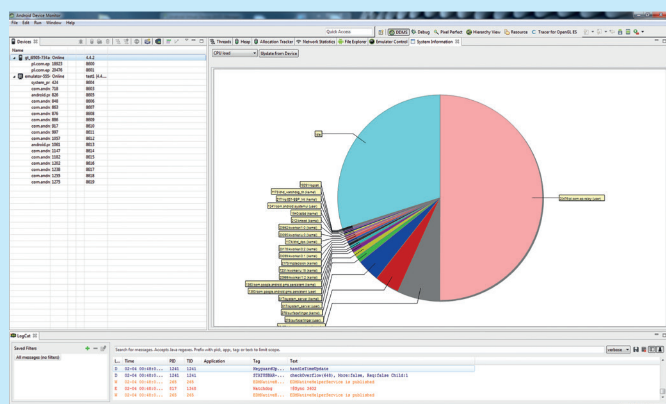


Rysunek 1. Android Device Monitor

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt



Rysunek 2. Pasek narzędziowy Android Device Monitor



Rysunek 3. Okno w ADM w perspektywie DDMS

Name			
gt_i9505-734a40c1	Online		4.4.2
pl.com.ep.relay	20476		8601 / 8700
pl.com.ep.domofon	32099		8600
emulator-5554	Online		test1 [4.4.4, debug]
system_process	424		8604
com.android.systemui	718		8603
android.process.acore	826		8605
com.android.inputmethod.latin	848		8606
com.android.phone	863		8607
com.android.launcher	876		8608
com.android.calendar	1147		8614
com.android.providers.calendar	1182		8615
com.android.mms	1202		8616
com.android.deskclock	1238		8617
com.android.email	1255		8618
com.android.dialer	1415		8610
com.android.browser	1431		8611
com.android.sharedstoragebackup	1449		8612

Rysunek 4. Lista podłączonych urządzeń i uruchomionych na nich procesów

zmienić przede wszystkim sposób prezentacji informacji, ale też wpływa na interfejsy używane do komunikacji ze sprzętem. Niemniej w praktyce, ustawienia domyślne będą wystarczające.

Główny pasek narzędzi programu (**rysunek 2**) służy do przełączania perspektyw oraz zawiera przyciski ułatwiające uruchomienie androidowego menedżera SDK i menedżera urządzeń wirtualnych.

DDMS

Podstawową perspektywą dla osób chcących debugować aplikacje androidowe jest DDMS, czyli Dalvik Debug Monitor Server. Słowo „Dalvik" to nazwa implementacji maszyny wirtualnej Javy, używanej w Androidzie. Dalvik przetwarza kod bajtowy, powstały przez kompilację kodu napisanego w języku Java. Natomiast biblioteki Cordovy dla Androida, napisane są właśnie w Javie i pozwalają uruchamiać kod napisany w JavaScripcie – przekierowują wywołania JavaScriptowe do odpowiednich komend i bibliotek Javy. W efekcie, z punktu widzenia maszyny wirtualnej, kod JavaScript, HTML i CSS to zasoby, z których korzysta kod aplikacji napisany w Javie i skompilowany do kodu bajtowego. Natomiast z punktu widzenia programisty tworzącego z użyciem Cordovy, właściwy kod jest napisany w JavaScripcie i odwołuje się do funkcji bibliotecznych, napisanych w Javie. Warto dodać, że w nowych wersjach Androida, tj. od 5.0 wzwyż, środowisko Dalvik zostało zupełnie zastąpione środowiskiem Android RunTime (ART), które nieco inaczej optymalizuje działanie aplikacji pod kątem konkretnego urządzenia, na którym jest ona instalowana. W praktyce jednak korzysta z tych samych plików kodu bajtowego i narzędzie DDMS może być używane do debugowania programów także na nowszym Androidzie.

DDMS przekazuje informacje pomiędzy komputerem, a podłączonym, debugowanym urządzeniem androidowym za pośrednictwem mostu ADB (Android Debug Bridge). Działa zarówno z fizycznymi smartfonami i tabletami, jak i z wirtualnymi. Ważne by na danym urządzeniu włączone było debugowanie przez USB (co opisywaliśmy w poprzedniej części kursu) oraz by testowana aplikacja była skompilowana w wersji przeznaczonej do debugowania. Nie stanowi to żadnego problemu, gdyż domyślnie kompilowane aplikacje w Cordovie są tworzone właśnie w takim trybie.

DDMS pozwala monitorować stan procesora, pamięci i stertę urządzenia. Umożliwia wykonywanie zrzutów ekranu, logowanie zdarzeń i komunikatów wyświetlanych w konsoli systemowej oraz symulowanie zdarzeń związanych ze stanem połączeń sieciowych, rozmowami przychodzącymi SMSami, pozorowaniem lokalizacji itp.

Widok całego okna DDMS pokazano na **rysunku 3**. Podstawowym elementem jest zakładka Devices, w której wymienione są podłączone urządzenia androidowe (realne i wirtualne) oraz uruchomione na nich aplikacje, które można debugować (**rysunek 4**). Jeśli podłączone urządzenie nie jest widoczne na tej liście, to znaczy, że prawdopodobnie wystąpił problem ze sterownikami. Ich instalację opisaliśmy w poprzedniej części kursu. Jeśli wcześniej wszystko wydawało się działać, ale po jakimś czasie przestało – przydatna okazuje się typowa strategia rozwiązywania problemów przez informatyków: należy zrestartować system, a jeśli to nie pomoże, ponownie przeinstalować sterowniki i podłączyć urządzenia na nowo.

Poprawnie podłączone i uruchomione urządzenie będzie miało status „Online" oraz podaną wersję zainstalowanego systemu operacyjnego. W przypadku urządzeń emulowanych podawana jest też nazwa wirtualnego sprzętu.

Każdemu z podłączonych urządzeń przypisana jest lista uruchomionych aplikacji, które można debugować. Warto zauważyć, że w przypadku symulatora, lista aplikacji dostępnych do debugowania jest znacznie większa, gdyż obejmuje także standardowe programy systemu Android. Obok odwrotnej nazwy domenowej każdej z aplikacji podany jest jej numer procesu PID (Process IDentification) i numer portu, wykorzystywanego do debugowania jej. Numery PID są unikalne w ramach danego urządzenia, a numery portów unikalne w ramach całego debugera. Poza tym w tabeli prezentowane są ikonki, symbolizujące operacje wykonywane przez debuger w związku z danym procesem.

Fakt, że każda aplikacja ma oddzielny port wynika ze specyficznego sposobu uruchamiania programów w systemie Android. Każdy proces w Androidzie ma

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hhcxxtt**

bowiem oddzielną maszynę wirtualną, w której pracuje. Każda maszyna natomiast komunikuje się z użyciem innego portu debugera. W efekcie, numery PID są w rzeczywistości numerami procesów maszyn wirtualnych, ale nie ma to większego znaczenia.

W momencie, gdy DDMS jest uruchamiane, łączy się ono z modułem Android Debug Bridge, który pozwala na komunikację z systemem Android. ADB to narzędzie tekstowe, które można obsługiwać i wywoływać ręcznie z linii poleceń. Wystarczy skorzystać z polecenia **adb.exe** – znajduje się ono w podkatalogu **platform-tools** katalogu z narzędziami deweloperskimi Androida. Jednakże samodzielne, ręczne używanie tej komendy zdecydowanie wykracza poza zakres tego kursu i raczej nie będzie potrzebne osobom tworzącym aplikacje Cordovy. Znacznie wygodniej jest używać graficznego interfejsu DDMS, który po uruchomieniu odpytuje poprzez ADB o numery PID poszczególnych procesów (ich maszyn wirtualnych) na danym urządzeniu i przypisuje im kolejne numery portów, przez które będzie się komunikować.

Podstawowe operacje

Nad listą urządzeń i uruchomionych na nich procesów znajduje się belka z przyciskami, przedstawiona na **rysunku 5**. Niektóre z wywoływanych przez nie poleceń są jednorazowe, a niektóre służą umożliwieniu wyświetlenia informacji w dalej opisywanych elementach DDMS.

Pierwszy z przycisków o kształcie żuka będzie zapewne wyszarzony, gdyż pozwala on na skakanie do odpowiednich miejsc kodu, gdy ten został skompilowany i uruchomiony bezpośrednio ze zintegrowanego środowiska deweloperskiego. Jeśli ktoś tworzy aplikacje Cordovy z użyciem Eclipse i z odpowiednimi ustawieniami, wtedy będzie mógł skorzystać z tej opcji. Jednakże w przeciwnym wypadku jest ona niedostępna. Kolejne dwa przyciski pomagają zarządzać pamięcią. Pierwszy służy do włączenia śledzenia sterty pamięci danej aplikacji. Drugi (ten z czerwoną strzałką) umożliwia zapis profilu sterty do pliku. Trzeci – kosz – uruchamia mechanizm czyszczenia pamięci po nieużywanych obiektach (*Garbage Collector*).

Następne dwa przyciski służą kontroli wątków wykonywanych w ramach wybranej aplikacji. Pierwszy włącza monitorowanie wątków, a drugi umożliwia dokładne śledzenie czasu ich wykonywania. Obie te funkcje zostały opisane w dalszej części artykułu.

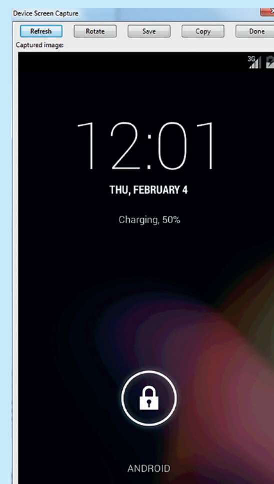
Przycisk „Stop Process" pozwala natychmiast zakończyć aplikację. Przycisk „Screen Capture" pozwala pobrać zrzut aktualnego ekranu z urządzenia mobilnego i zapisać go (**rysunek 6**). Przycisk „Dump View Hierarchy for UI Automator" pozwala natomiast analizować elementy interfejsu graficznego (**rysunek 7**). Z punktu widzenia programisty tworzącego w Cordovie jest on jednak znacznie mniej przydatny niż w aplikacjach pisanych w natywnym języku. Cała wyświetlana treść jest traktowana jako jedno okno i nie można wnikać w jej szczegóły, jak to było przy debugowaniu kodu HTML, opisywanego w poprzedniej części kursu. Widok ten jest przydatny w aplikacjach Cordovy tylko wtedy, gdy program nie zajmuje całego ekranu, na przykład, gdy na jego górze znajduje się pasek stanu. Wtedy w widoku hierarchii da się rozróżnić te dwa elementy.

Rysunek 5. Pasek narzędzi nad listą urządzeń

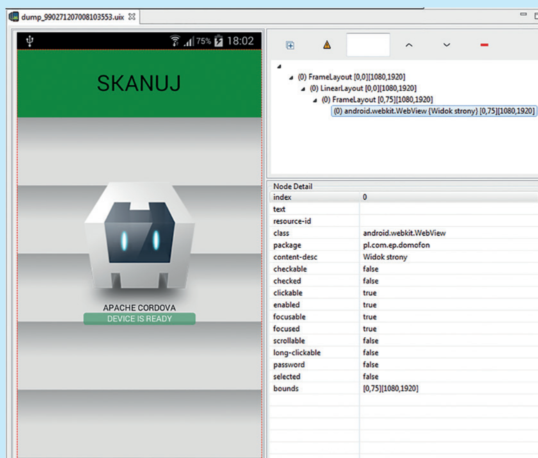
Ponadto widok pokazuje listę parametrów okna, a w tym m.in. jego rozmiary.

Przycisk „Capture system wide trace using android systrace" pozwala zebrać przez określony czas wszystkie informacje o zdarzeniach zachodzących w urządzeniu mobilnym lub w wybranej aplikacji i zapisać je do pliku (**rysunek 8**) – można też szczegółowo określić opcje śledzenia. Komenda Start OpenGL Trace (**rysunek 9**) umożliwia natomiast śledzenie wywołań graficznych OpenGL i przeglądanie zdarzeń z nimi związanych klatka po klatce (**rysunek 10**).

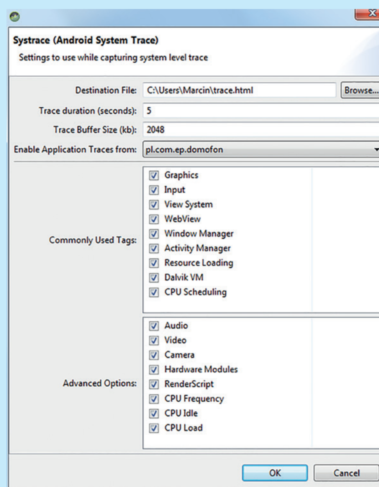
W końcu, pod strzałką, w rozwijanym menu, oprócz wcześniej wymienionych opcji znajduje się też przycisk resetowania Android Debug Bridge. Powoduje ona rozłączenie wszystkich dotychczasowych



Rysunek 6. Wynik pracy narzędzia wykonującego zrzuty ekranu z podłączonych urządzeń. Zrzuty są w pełnej rozdzielczości, w jakiej pracuje dane urządzenie



Rysunek 7. Podgląd warstw składających się na wyświetlane okno



Rysunek 8. Narzędzie do zapisywania do pliku informacji o zdarzeniach w ramach aplikacji, zgodnie z ustawionymi filtrami

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hcxxtt

Nowości w Cordovie

Od czasu powstania 8 części niniejszego kursu wprowadzono kolejne zmiany w platformie Cordova.
Przede wszystkim aktualna wersja platformy Cordova i jej bibliotek to 6.0.0. Korzysta ona z modułu plugman 1.1.0 i cordova-js 4.1.3. Podstawowe zmiany obejmują zwiększenie domyślnych wersji bibliotek poszczególnych platform sprzętowych:

- cordova-android do wersji 5,
- cordova-ios do wersji 4,
- cordova-windows do wersji 4.3.

Platforma cordova-ios w wersji 4.0 wspiera iOS9 i WKWebView oraz wprowadza wiele usprawnień w zakresie wyświetlania stron w internetowych, konfiguracji ikon i ekranu startowego, natomiast opcjonalna i niedawno wprowadzona wersja 5.1 platformy cordova-android obsługuje Androida 6.x.x, czyli Marshmallow.
W Cordovie 6.0.0. wprowadzono nową funkcję tworzenia projektów w oparciu o szablony pobierane np. z Internetu (opcja --template). Wycofano natomiast wsparcie dla starego rejestru wtyczek Cordovy – obecnie obsługiwane są tylko te w zasobach npm, git lub ze ścieżek lokalnych.
Wprowadzono też ustawienie, sprawiające że domyślnie w trakcie pobierania standardowych wtyczek platforma będzie sięgać po te wersje, które były jej przypisane w trakcie jej utworzenia, nawet jeśli dostępne będą już nowsze aktualizacje. Pobranie nowszej wersji wtyczki będzie wymagało ręcznego wyboru ze strony użytkownika.
Wersje platform i wtyczek, przypisanych do Cordovy 6.0.0 są następujące:

- Cordova Amazon-FireOS: 3.6.3
- Cordova Android: 5.1.0
- Cordova BlackBerry10: 3.8.0
- Cordova Browser: 4.0.0
- Cordova FirefoxOS: 3.6.3
- Cordova iOS: 4.0.1
- Cordova OSX: 4.0.0
- Cordova Ubuntu: 4.3.2
- Cordova Windows: 4.3.0
- Cordova WebOS: 3.7.0
- Cordova WP8: 3.8.2
- cordova-plugin-battery-status: 1.1.1
- cordova-plugin-camera: 2.1.0
- cordova-plugin-console: 1.0.2
- cordova-plugin-contacts: 2.0.1
- cordova-plugin-device: 1.1.1
- cordova-plugin-device-motion: 1.2.0
- cordova-plugin-device-orientation: 1.0.2
- cordova-plugin-dialogs: 1.2.0
- cordova-plugin-file: 4.1.0
- cordova-plugin-file-transfer: 1.5.0
- cordova-plugin-geolocation: 2.1.0
- cordova-plugin-globalization: 1.0.2
- cordova-plugin-inappbrowser: 1.2.0
- cordova-plugin-legacy-whitelist: 1.1.1
- cordova-plugin-media: 2.1.0
- cordova-plugin-media-capture: 1.2.0
- cordova-plugin-network-information: 1.2.0
- cordova-plugin-splashscreen: 3.1.0
- cordova-plugin-statusbar: 2.1.0
- cordova-plugin-test-framework: 1.1.1
- cordova-plugin-vibration: 2.1.0
- cordova-plugin-whitelist: 1.2.1

Niektóre z tych wtyczek zostały zaktualizowane, by korzystać z nowości w platformie cordova-android 5.1.0, które pozwalają uniknąć błędów, jakie dotąd powstawały, gdy system operacyjny, w celu uwolnienia zasobów, czyścił pamięć i m.in. usuwał oczekujące wywołania zwrotne funkcji.
Platforma cordova-windows 4.3 wspiera niektóre nowe metody wywoływania i logowania operacji, a platforma cordova-ubuntu 4.3.2 zawiera jedynie drobne aktualizacje.
Wprowadzono też komunikat o zbliżającym się wycofaniu wsparcia dla platform amazon-fireos i wp8. Ma to nastąpić za ok. 6 miesięcy i zaleca się korzystanie zamiast nich z platform Android i Windows. Warto dodać, że cordova-ios w wersji 4.0.0. i 4.0.1 (a więc w opcjonalnej aktualizacji) pozwala na kompilację na urządzenia z systemem nie starszym niż iOS 8.0.

Rysunek 9. Narzędzie do śledzenia akcji OpenGL

ID	Tid	Status	utime	stime	Name
1	24540	Native	31217	4001	main
*2	24544	VmWait	0	4	GC
*3	24545	VmWait	0	0	Signal Catcher
*4	24546	Runnable	17	64	JDWP
*5	24547	VmWait	27	26	Compiler
*6	24548	Wait	0	0	ReferenceQueueDaemon
*7	24549	Wait	0	1	FinalizerDaemon
*8	24550	Wait	0	0	FinalizerWatchdogDaemon
9	24551	Native	203	103	Binder_1
10	24552	Native	165	107	Binder_2
11	24560	Native	95	64	Thread-13474
*12	24566	Wait	0	0	CleanupReference
13	24555	Native	0	0	Thread-13477
14	26526	Wait	0	0	AsyncTask #2
15	24572	Native	1	2	JavaBridge
16	24569	Native	423	654	Thread-13480
17	24580	Native	182	108	Binder_3
18	24561	Native	1	0	Thread-13482
19	24581	Native	384	180	Thread-13483
20	24583	Native	0	1	Thread-13484
21	24584	Native	1	0	Thread-13485
22	26532	Wait	0	0	AsyncTask #3
23	24602	Wait	0	0	AsyncTask #1
24	26629	Wait	0	0	AsyncTask #4
25	26635	TimedWait	0	0	AsyncTask #5
26	26658	Runnable	5812	162	Thread-13495

```
Refresh  Thu Feb 04 13:11:47 CET 2016
at com.google.zxing.common.HybridBinarizer.thresholdBlock(HybridBinarizer.java:147)
at com.google.zxing.common.HybridBinarizer.calculateThresholdForBlock(HybridBinarizer.ja...
at com.google.zxing.common.HybridBinarizer.getBlackMatrix(HybridBinarizer.java:82)
at com.google.zxing.BinaryBitmap.getBlackMatrix(BinaryBitmap.java:83)
at com.google.zxing.qrcode.QRCodeReader.decode(QRCodeReader.java:76)
at com.google.zxing.MultiFormatReader.decodeInternal(MultiFormatReader.java:170)
at com.google.zxing.MultiFormatReader.decodeWithState(MultiFormatReader.java:85)
at com.google.zxing.client.android.DecodeHandler.decode(DecodeHandler.java:82)
at com.google.zxing.client.android.DecodeHandler.handleMessage(DecodeHandler.java:60)
at android.os.Handler.dispatchMessage(Handler.java:102)
at android.os.Looper.loop(Looper.java:157)
at com.google.zxing.client.android.DecodeThread.run(DecodeThread.java:94)
```

Rysunek 10. Lista akcji OpenGL, śledzona klatka po klatce

połączeń debugera i załadowanie oraz przypisanie im portów na nowo.

Podgląd wątków

Każdy proces może mieć wiele wątków. Każdy wątek w DDMS ma dwa identyfikatory – jeden unikalny w ramach danej aplikacji (ID), a jeden unikalny w ramach całego urządzenia (Tid). Numer pierwszego, podstawowego wątku (main) pokrywa się z numerem PID uruchomionej aplikacji. Podany jest też stan wątku, jego nazwa oraz czas wykorzystania procesora, podzielony na dwie kategorie: *utime* i *stime* (**rysunek 11**). Pierwsza z nich, *utime*, obejmuje czas

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hhcxxtt**

ID	Tid	Status	utime	stime	Name
1	24540	Native	31217	4001	main
*2	24544	VmWait	0	4	GC
*3	24545	VmWait	0	0	Signal Catcher
*4	24546	Runnable	17	64	JDWP
*5	24547	VmWait	27	26	Compiler
*6	24548	Wait	0	0	ReferenceQueueDaemon
*7	24549	Wait	0	1	FinalizerDaemon
*8	24550	Wait	0	0	FinalizerWatchdogDaemon
9	24551	Native	203	103	Binder_1
10	24552	Native	165	107	Binder_2
11	24560	Native	95	64	Thread-13474
*12	24566	Wait	0	0	CleanupReference
13	24555	Native	0	0	Thread-13477
14	26526	Wait	0	0	AsyncTask #2
15	24572	Native	1	2	JavaBridge
16	24569	Native	423	654	Thread-13480
17	24580	Native	182	108	Binder_3
18	24561	Native	1	0	Thread-13482
19	24581	Native	384	180	Thread-13483
20	24583	Native	0	1	Thread-13484
21	24584	Native	1	0	Thread-13485
22	26532	Wait	0	0	AsyncTask #3
23	24602	Wait	0	0	AsyncTask #1
24	26629	Wait	0	0	AsyncTask #4
25	26635	TimedWait	0	0	AsyncTask #5
26	26658	Runnable	5812	162	Thread-13495

Threads | Heap | Allocation Tracker | Network Statistics | File Explorer | Emul

Refresh Thu Feb 04 13:11:47 CET 2016

```
at com.google.zxing.common.HybridBinarizer.thresholdBlock(HybridBinarizer.java:147)
at com.google.zxing.common.HybridBinarizer.calculateThresholdForBlock(HybridBinarizer.ja...
at com.google.zxing.common.HybridBinarizer.getBlackMatrix(HybridBinarizer.java:82)
at com.google.zxing.BinaryBitmap.getBlackMatrix(BinaryBitmap.java:83)
at com.google.zxing.qrcode.QRCodeReader.decode(QRCodeReader.java:76)
at com.google.zxing.MultiFormatReader.decodeInternal(MultiFormatReader.java:170)
at com.google.zxing.MultiFormatReader.decodeWithState(MultiFormatReader.java:85)
at com.google.zxing.client.android.DecodeHandler.decode(DecodeHandler.java:82)
at com.google.zxing.client.android.DecodeHandler.handleMessage(DecodeHandler.java:60)
at android.os.Handler.dispatchMessage(Handler.java:102)
at android.os.Looper.loop(Looper.java:157)
at com.google.zxing.client.android.DecodeThread.run(DecodeThread.java:94)
```

Rysunek 11. Podgląd wątków w aplikacji

procesora użyty do wykonywania kodu użytkownika. Drugi – *stime*, to czas, który dany wątek wykorzystał na żądania usług systemowych. Obie wartości są podane w jednostkach „jiffie", których dokładna długość zależna jest od ustawień systemowych, ale zazwyczaj odpowiada 10 ms. Wątki o identyfikatorach (ID) z gwiazdką to demony systemu Linux, a więc programy działające w tle, trochę tak jak usługi w Windows. Natomiast stan wątku może przybierać jedną z wartości podanych w **tabeli 1**. Należy przy tym zaznaczyć, że wątki zakończone są usuwane z listy. Natomiast nazwy wątków mogą być pomocne w określeniu tego, za którą operację odpowiadają. Wywołania zewnętrznych funkcji urządzenia mobilnego, wykonywane z poziomu Cordovy często są uruchamiane asynchronicznie z nazwą AsyncTask lub Thread-. Warto też orientować się, że wątek GC odpowiada mechanizmowi czyszczenia pamięci po niepotrzebnych obiektach (Garbage Collector), a wątek JDWP (Java Debug Wire Protocol) służy do komunikacji z ADB, a więc i DDMS.

Zaznaczając wybrany wątek z listy otrzymujemy podgląd aktualnych ścieżek wywołania dla niego. Na **rysunku 11** zaznaczono wątek odpowiadający wywołaniu skanera kodów QR, użytego w przykładzie aplikacji domofonu, opisanego w jednej z poprzednich części tego kursu. Jeśli nie mamy pewności, który wątek odpowiada za interesujące nas zadanie, możemy przeglądać listy wywołania w poszukiwaniu rozpoznawalnych nazw bibliotek, które mogą być powiązane z używanymi wtyczkami Cordovy. Warto klikać przycisk „refresh", gdyż aktualne ścieżki wywołania stale się zmieniają dla wielu wątków.

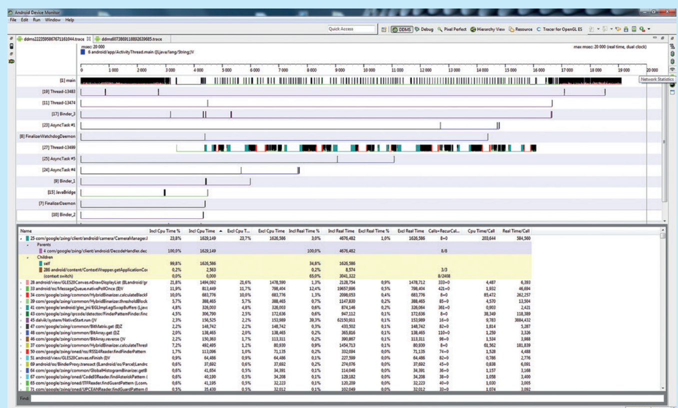
Tabela 1. Dopuszczalne stany wątków w aplikacji androidowej w DDMS

Stan	Opis
INITIALIZING	Wątek, który ma być dopiero uruchomiony
STARTING	Wątek, który jest właśnie uruchamiany, ale jeszcze nie został w pełni uruchomiony
ZOMBIE	Wątek, który właśnie jest zamykany
RUNNING	Wątek uruchomiony, aktywnie obsługiwany w danej chwili przez procesor
WAIT	Wątek wstrzymany, czekający na wzbudzenie
TIMED_WAIT	Wątek wstrzymany z określonym z góry momentem wzbudzenia
MONITOR	Wątek zablokowany, oczekujący na jakieś konkretne zdarzenie – np. w ramach synchronizacji z innym wątkiem
NATIVE	Wątek wykonywany w kodzie natywnym. Standardowo nie jest on zawieszany, chyba że wywoła wątek Javy, co powoduje przełączenie jego stanu do RUNNING, a następnie błyskawicznie do SUSPENDED
VMWAIT	Wątek wstrzymany ze względu na operacje wykonywane przez maszynę wirtualną – np. przez mechanizm Garbage Collection lub w oczekiwaniu na zasoby udostępniane przez maszynę wirtualną
SUSPENDED	Wątek zawieszony, co ma miejsce np. wtedy, gdy maszyna wirtualna wykonuje operacje na jego stosie.

Profilowanie wątków

Jeśli problem z aplikacją leży w jej wydajności, można sięgnąć po narzędzie do profilowania, dostępne w DDMS. Uruchamia się je i zatrzymuje ręcznie, po czym prezentowany jest raport. Dostępne są dwa tryby monitorowania wykonania wątków: zgrubny, w którym debuger co jakiś czas odpytuje procesor o to, które wątki są uruchomione oraz dokładny, który ciągle monitoruje wykonanie programu w oparciu o wywołania i zakończenia poszczególnych metod. Pierwszy tryb jest mniej obciążający procesor – można określić częstość odpytywań z dokładnością do jednej mikrosekundy (domyślna wartość to 1000 mikrosekund). Drugi, zdecydowanie bardziej ogranicza procesor i wydaje się, że wygodniejsze jest korzystanie z pierwszego trybu.

Gotowe raporty otwierają się w nowych zakładkach i składają z dwóch segmentów: wykresu określającego przebieg wykonywania wątków w czasie oraz tabelarycznego podsumowania czasu poświęconego na poszczególne wątki (**rysunek 12**). Po raporcie można nawigować myszką, wnikając w szczegóły. Najechanie kursorem na dany wątek na wykresie informuje,



Rysunek 12. Wynik śledzenia wykonywanych wątków w czasie

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

GPX i KML

Formaty GPX i KML służą do przechowywania tras i ścieżek opartych o koordynaty geograficzne, pozyskiwane najczęściej z użyciem nawigacji satelitarnej – głównie GPS, a ostatnio także GLONASS.
Format GPX (GPS Exchange Format) obejmuje schematy XML, w których ramach zapisywane są ścieżki, trasy i pojedyncze punkty na ziemi. Typowo dane zawierają informacje o długości i szerokości geograficznej, a także o wysokości nad poziomem morza i o czasie. Pliki te można wygenerować ręcznie, z użyciem oprogramowania lub za pomocą urządzeń do nawigacji – np. jako zarejestrowana, przebyta trasa.
Format KML (Keyhole Markup Language) również obejmuje schematy XML. Pierwotnie został opracowany przez firmę Keyhole, twórcę programu, który obecnie (wraz z firmą) należy do Google i nazywa się Google Earth. Format KML zawiera różnego rodzaju obiekty, które można prezentować na dwu- i trójwymiarowych mapach. Przykładowo, można w nim zapisać punkty orientacyjne, wraz z ich nazwami, koordynatami i opisami. Istnieją konwertery przetwarzające pliki pomiędzy obydwoma formatami. Jednym z nich, dostępnym online jest http://gpx2kml.com/.

co dokładnie było wykonywane w danej chwili – podawany jest numer wątku wywołanego w ramach danego wątku, jego ścieżka wywołania oraz informacje o wykorzystanym czasie. Ponieważ wykres prezentowany jest z dokładnością do mikrosekund, można go powiększać poprzez zaznaczenie interesującego nas obszaru. Powrót do pełnej skali wymaga podwójnego kliknięcia lewym przyciskiem myszy w górnej części wykresu. Pojedyncze kliknięcie w obszarze wątków rozwinie natomiast listę wywołań danego wątku w tabelce poniżej wykresu.

W tabeli, dla czytelności, poszczególne wątki są oznaczane różnymi kolorami – kolory te nie mają większego znaczenia (jedynie czasem istotne są powiązania pomiędzy wątkami o tym samym kolorze). w Hierarchicznej postaci prezentowane jest, co wywołało dany wątek (*Parents*) i co dany wątek wywołał sam (*Children*) kliknięcie na dowolny wątek w ramach tych kategorii błyskawicznie przenosi nas do fragmentu tabeli z podglądem szczegółów dla właśnie niego. Przykładowo, wybierając dowolny wątek z listy i klikając zawsze na wątek z kategorii *Parents*, dotrzemy do wątku zerowego (*toplevel*).

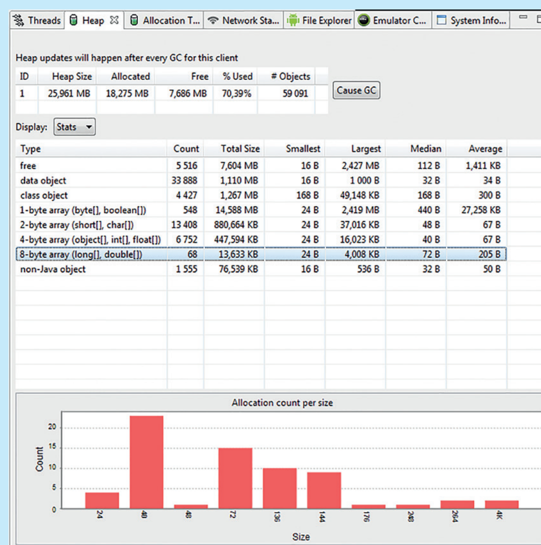
Podane w tabeli czasy są zapisane w mikrosekundach oraz w procentach, a także zostały podzielone na cztery kategorie: czas łączny procesora, czas wyłączny procesora oraz czas łączny rzeczywisty i czas wyłączny rzeczywisty. Czasy wyłączne obejmuje tylko te okresy, gdy funkcje faktycznie działa i wykonuje operacje, podczas gdy czasy łączne, zawierają też okresy, w których wątek ma przestój – nakazano mu czekać na wykonanie innej funkcji. Czasy rzeczywiste różnią się od czasu procesora tym, że obejmują również czas potrzebny na urządzenia wejścia i wyjścia, na które procesor musi czekać. W efekcie, wątek 0 (*toplevel*), choć zajmuje bardzo dużo czasu łącznego procesora, właściwie wcale nie zajmuje procesora na wyłączność (czas na wyłączność wątku 0 widać praktycznie tylko przy bardzo dokładnym profilowaniu, np. z użyciem drugiego z wymienionych wcześniej trybów).

Warto też ewentualnie zwrócić uwagę na prawą stronę tabeli, gdzie znajduje się liczba wywołań funkcji oraz czas procesora i czas rzeczywisty, w przeliczeniu na pojedyncze wywołanie. Pozwalają one lepiej ocenić czas potrzebny na jeden przebieg konkretnej funkcji.

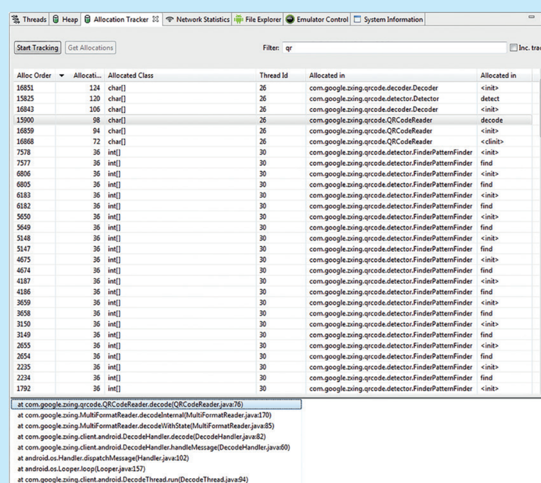
Monitorowanie sterty pamięci

Jeśli aplikacja wiesza się z czasem, bardzo możliwe, że następują jakieś wycieki pamięci. Da się wykryć ich istnienie poprzez monitorowanie sterty (kopca) pamięci. Informacje na ten temat zawiera zakładka „Heap", po włączeniu aktualizacji danych sterty. Dane aktualizowane są za każdym razem, gdy uruchamiany jest mechanizm czyszczenia niepotrzebnych obiektów z pamięci (*Garbage Collector*), przy czym dzieje się to dosyć nieregularnie, więc w zakładce poświęconej stercie umieszczono też przycisk wymuszający włączenie GC (*Cause GC*).

W zakładce „Heap" można dowiedzieć się, jaki jest łączny rozmiar pamięci, zarezerwowanej dla aplikacji, ile z tej pamięci jest aktualnie w użyciu, a ile wolnej, jaki to procent całej zarezerwowanej oraz ile obiektów składa się na tę zajętą przestrzeń. Poniżej prezentowany jest widok szczegółowy, w którym pokazane są rodzaje obiektów, takie jak klasy i tablice zawierające pola o różnych wielkościach. Dla każdej z kategorii wyświetlona jest liczba elementów danego typu, łączna zajmowana przestrzeń oraz statystyki: najmniejsza wielkość obiektu, największa, mediana i wartość średnia. Wybranie



Rysunek 13. Analiza zajętości sterty pamięci



Rysunek 14. Śledzenie alokacji pamięci

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hlcxxtt

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Tabela 2. Opcje programisty w systemie Android 4.4 w telefonie Samsung Galaxy S4

Opcja PL	Opcja EN	Wyjaśnienie
		Podstawowe
Utwórz raport o błędach	Take bug report	Pozwala przygotować raport o błędach, które wystąpiły w trakcie działania dowolnej aplikacji. Trwa to maksymalnie kilka minut i raport udostępniany jest do wysyłki pocztą, za pomocą wybranego programu pocztowego
Hasło backupu urządzenia stacjonarnego	Desktop backup password	Pozwala wprowadzić ustawienia haseł pełnego backupu wykonywanego na komputerze
Pozostań w stanie czuwania	Stay awake	Pozwala zablokować usypianie telefonu, jeśli ten jest podłączony do zasilania
Log Bluetooth HCI snoop	Enable Bluetooth HCI snoop log	Opcja ta sprawia, że cała komunikacja z użyciem pakietów HCI (Host Controller Interface) Bluetooth będzie zapisywana do pliku /sdcard/btsnoop_hci.log i można ją będzie później obejrzeć w celach diagnostycznych
Statystyki procesów	Process stats	Wyświetla informacje o wykorzystaniu procesora i pamięci poszczególnych aplikacji i ich elementów
		Debugowanie
Debugowanie USB	USB debugging	Umożliwia korzystanie z omówionych w tej i poprzedniej części kursu narzędzi debugujących przez USB
Unieważnij autoryzację debugow. USB	Revoke USB debugging authorizations	Po podłączeniu nowego urządzenia debugującego do telefonu, na ekranie pojawia się pytanie, czy danemu urządzeniu należy zezwolić na debugowanie. Taka zgoda jest następnie zapisywana, by nie pytać ponownie. Niniejsza opcja pozwala wyczyścić wszystkie dotychczas wydane zgody.
Umieszczaj raporty o błędach w menu zasilania	Power menu bug reports	Dodaje opcję utworzenia raportu o błędach w menu zasilania, otwieranym przyciskiem włączania i wyłączania telefonu (rysunek 24)
Pozorowane położenie	Allow mock locations	Pozwala symulować położenie, np. z użyciem specjalnych aplikacji do tego służących. Niektóre aplikacje sprawdzają, czy opcja nie jest włączona i w przeciwnym razie nie pozwalają się uruchomić
Wybierz aplikację do debugowania	Select debug app	Pozwala wybrać, która z aplikacji będzie podlegać debugowaniu. Opcja przydatna, gdy w tle działają inne aplikacje, które mogą być debugowane
Poczekaj na debugera	Wait for debugger	Jeśli wybrano aplikację do debugowania, opcja ta sprawi, że wskazany program będzie czekał z uruchomieniem do momentu podłączenia debugera
Sprawdzaj aplikacje przez USB	Verify apps over USB	Wymusza skanowanie aplikacji instalowanych z poziomu debugera pod kątem złośliwego oprogramowania
		Wprowadź tekst
Pokaż dotknięcia	Show touches	Sprawia, że punkty dotknięcia ekranu dotykowego są oznaczane jasno-szarą kropką
Pokaż lokal. wskaźnika	Pointer location	Wyświetla na pasku na górze ekranu szczegółowe informacje na temat punktów dotknięcia oraz przesunięć palcem po ekranie
		Rysowanie
Pokaż aktualizacje ekranu	Show surface updates	Sprawia, że za każdym razem, gdy wyświetlana treść na ekranie się zmienia, ekran dodatkowo miga na fioletowo lub jest otaczany wąską czerwoną ramką
Pokaż granice układu	Show layout bounds	Pokazuje prostokątne granice elementów wyświetlanych na ekranie
Wymuś układ RTL	Force RTL layout direction	Wymusza wyrównanie układu ekranu do prawej strony we wszystkich ustawieniach lokalnych na potrzeby obsługi języków czytanych od prawej do lewej

Tabela 2. cd.

Opcja PL	Opcja EN	Wyjaśnienie
Skala animacji okna	Window animation scale	Pozwala zmienić szybkość systemowej animacji okien, a nawet ją wyłączyć
Skala animacji przejścia	Transition animation scale	Pozwala zmieniać szybkość przejść animacji, a nawet je wyłączyć
Skala długości animacji	Animation duration scale	Zmienia czas trwania animacji (wymagany restart urządzenia)
Symuluj dod. urządz. wizyj.	Simulate secondary displays	Pokazuje dodatkowe okno o symulowanej rozdzielczości, w którym wyświetlana jest aktualna treść ekranu
Renderowanie z przyspieszeniem sprzętowym		
Wymuś rendering GPU	Force GPU rendering	Pozwala wymusić korzystanie ze sprzętowej akceleracji grafiki 2D
Wyświetl aktualizacje widoku GPU	Show GPU view updates	Zaznacza zmieniające się właśnie fragmenty grafiki 2D, rysowane z użyciem procesora graficznego
Wyśw. aktual. warstw sprzęt.	Show hardware layer updates	Kolorem zielony oznacza warstwy grafiki w trybie akceleracji 2D, które ulegają aktualnie zmianom
Debuguj przerysowanie GPU	Debug GPU overdraw	Pozwala analizować, kiedy system nakazuje aplikacji przerysowanie fragmentów ekranu w trybie akceleracji 2D
Wyświetlanie nieprostokątnych przycięć	Debug non-rectangular clip operations	Ułatwia rozpoznawanie obszarów i obiektów nieprostokątnych
Włączanie 4x MSAA	Force 4x MSAA	Wymusza anti-aliasing w aplikacjach korzystających z OpenGL ES 2.0
Wyłącz nakładki sprzętu	Disable HW overlays	Wymusza korzystanie z procesora graficznego do tworzenia kompozycji ekranu
Monitorowanie		
Tryb ścisły	Strict mode enabled	Opcja powoduje miganie ekranu, gdy dana aplikacja wykonuje czasochłonne operacje w głównym wątku
Pokaż wykorz. procesora	Show CPU usage	Pokazuje na ekranie dodatkową warstwę z informacjami o wykorzystaniu procesora poprzez poszczególne procesy
Profil renderingu GPU	Profile GPU rendering	Pokazuje przebieg wykorzystania procesora graficznego w czasie - dostępne są różne sposoby prezentacji tych informacji
Włącz ślady OpenGL	Enable OpenGL traces	Pozwala określić, w jaki sposób mają być przekazywane informacje o pracy bibliotek OpenGL
Aplikacje		
Wył. aplikacje w tle	Don't keep activities	Powoduje automatyczne zamykanie aplikacji, po wyjściu z nich, co sprawia, że nie mogą działać w tle
Ogranicz procesy w tle	Background process limit	Pozwala ograniczyć liczbę procesów działających w tle
Pok. niestabilne aplik.	Show all ANRs	Sprawia, że aplikacje, które się zawieszają i nie odpowiadają, wyświetlają okno informujące o tym nawet, jeśli dany proces działa w tle

dowolnego z wierszy w tej tabeli powoduje wyświetlenie histogramu poszczególnych obiektów danego typu, a więc liczby zaalokowanych obiektów o konkretnym rozmiarze. Zademonstrowano to na **rysunku 13**.

Zajmowaną pamięć można też dokładniej obserwować dzięki mechanizmowi śledzenia alokacji. W tym celu używa się zakładki „Allocation Tracker". Na jej górze znajdują się dwa przyciski. Pierwszy włącza i wyłącza monitorowanie, a przycisk „Get Allocations" pozwala na pobranie aktualnej listy alokacji, w trakcie, gdy śledzenie jest włączone. Ponieważ zajmowanie pamięci dla nowych obiektów odbywa się praktycznie non stop, powstała tabela z listą operacji tego typu już po chwili staje się ogromna. Na szczęście można

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

ją sortować, a przede wszystkim filtrować. Pozycje w tabeli obejmują numer porządkowy alokacji, wielkość zarezerwowanej pamięci, rodzaj alokowanego obiektu (np. liczba typu int, albo wskazanie miejsca definicji klasy obiektu), numer wątku odpowiedzialnego za alokację oraz klasę i funkcję, w której alokacja została dokonana. Kliknięcie na pozycję w tej tabeli spowoduje wyświetlenie ścieżki wywołania danej funkcji. Pokazano to na **rysunku 14**.

Monitorowanie komunikacji

Kolejnym elementem udostępnianym przez DDMS jest monitorowanie komunikacji z urządzeniem. W przypadku emulatorów możliwości jest więcej niż dla realnych urządzeń. Prawdziwy telefon pozwala na podgląd ilości danych przesyłanych przez sieć – służy temu zakładka „Network Statistics". Zawiera ona wykres oraz tabelkę. Na wykresie prezentowana jest wykorzystywana przepustowość wysyłanych i pobieranych danych w czasie. Wykres może być odświeżany z różną częstością (co 100, 250 lub 500 ms), a proces ten zaczyna się od momentu wciśnięcia przycisku „Start". Przycisk „Reset" pozwala wyczyścić aktualnie zebrane dane i przez to zzerować statystyki zebrane w tabeli poniżej. Tabela wskazuje liczby danych przesłanych i odebranych, liczone w bajtach i w pakietach (**rysunek 15**).

Znacznie ciekawsze możliwości oferuje zakładka „Emulator Control", która umożliwia symulowanie zjawisk zewnętrznych, dotyczących telefonu (**rysunek 16**). Przede wszystkim możliwe jest zmienienie stanu dostępności sieci dla połączeń głosowych i danych. Dostępne opcje to:

- **Unregistered**, gdy telefon nie połączył się z żadną siecią, ponieważ np. nie ma włożonej karty SIM.
- **Home**, gdy telefon jest połączony z siecią macierzystą.
- **Roaming**, gdy telefon jest połączony z siecią inną, niż macierzysta (przypisana do włożonej karty SIM).
- **Searching**, gdy telefon poszukuje sieci.
- **Denied**, gdy dostęp do sieci jest zablokowany.

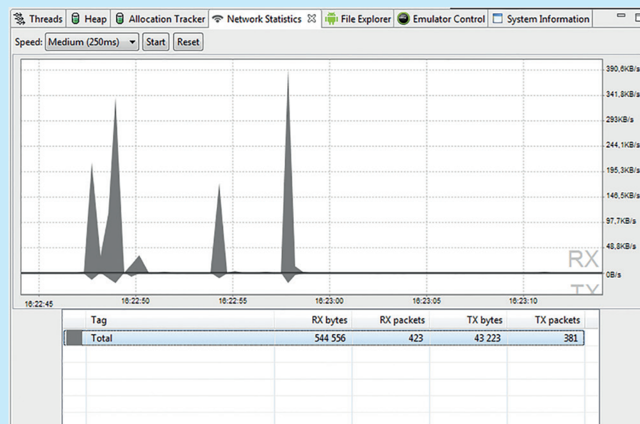
Można też określić szybkość połączenia. Dostępne opcje to: Full, GSM, HSCSD, GPRS, EDGE, UMTS i HSDPA. Lista wyboru opóźnień w transmisji zawiera pozycje: None, GPRS, EDGE i UMTS.

Poniżej znajduje się narzędzie do symulowania połączenia przychodzącego lub wiadomości. Wystarczy wpisać numer telefonu nadawcy i – jeśli ma to być wiadomość SMS – wpisać jej treść oraz nacisnąć przycisk wywołania. Efekt został przedstawiony na **rysunkach 17** i **18**.

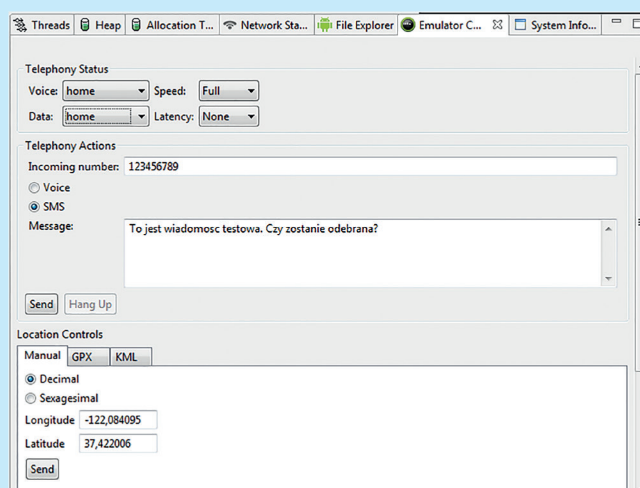
Przy budowie aplikacji, w której system korzysta z danych GPS można je symulować wpisując ręcznie w dziale Location Controls. Dostępne są dwie formy zapisu (dziesiętna oraz w postaci stopni, minut i sekund). Alternatywnie dane odnośnie lokalizacji albo ich szeregu można wprowadzić z plików GPX lub KML (patrz ramka o GPX i KML). Dzięki temu można odtworzyć raz zaprogramowaną ścieżkę, a nawet zmieniać szybkość poruszania się po niej.

Podglądanie zasobów

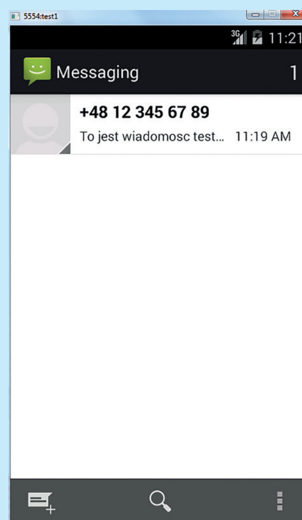
DDMS obejmuje jeszcze dwie interesujące zakładki. Jedna z nich to „File Explorer", który pozwala na wniknięcie w strukturę katalogową urządzenia mobilnego (**rysunek 19**). Jest to bardzo przydatne, gdy tworzona aplikacja zapisuje dane do pamięci trwałej lub gdy z nich korzysta. „File Explorer" pozwala na przeglądanie drzewa katalogów systemu plików, podając nazwy, ścieżki, rozmiary, prawa dostępu i informacje



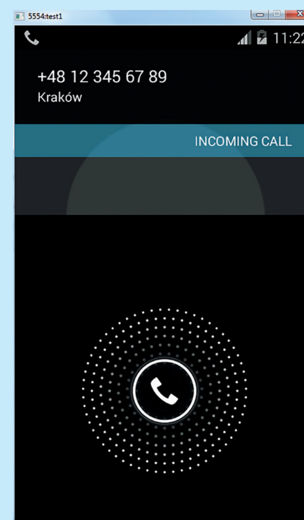
Rysunek 15. Podgląd ruchu sieciowego w urządzeniu



Rysunek 16. Narzędzie do sterowania emulatorem, umożliwiające symulowanie różnych warunków dostępu do sieci komórkowej, połączenia i wiadomości przychodzące oraz wskazania nawigacji satelitarnej



Rysunek 17. Widok wiadomości wysłanej z zakładki „Emulator Control" do emulowanego urządzenia mobilnego

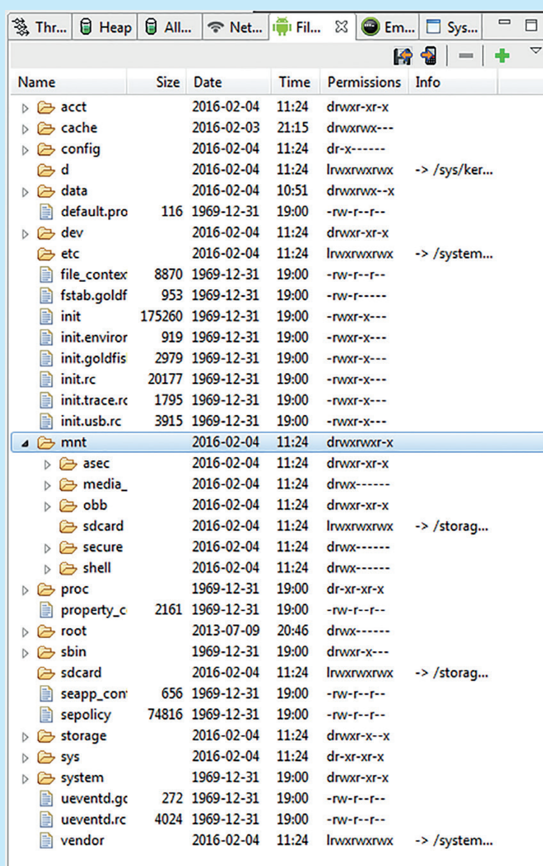


Rysunek 18. Symulowane połączenie przychodzące

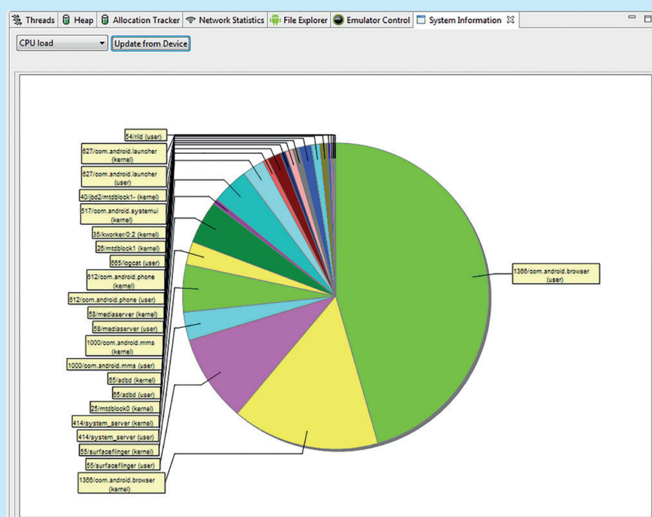
Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

o aliasach oraz uprawnieniach poszczególnych plików i katalogów. Można w trakcie pracy aplikacji stworzyć nowy katalog, usunąć plik, pobrać plik z urządzenia mobilnego na dysk komputera oraz przesłać plik z komputera w konkretne miejsce systemu plików urządzenia mobilnego.

Drugą zakładką jest „System Information”, która wskazuje podział zasobów pomiędzy poszczególne procesy oraz zagospodarowanie pamięci urządzenia mobilnego. Wybierając z listy pozycję CPU load i klikając „Update from Device” otrzymujemy wykres kołowy wskazujący, które procesy w ostatnim czasie zużywały jaką część czasu procesora (**rysunek 20**). Najechanie na fragmenty wykresu pozwala poznać szczegółowe dane. Na **rysunku 21** pokazano natomiast jak wyglądają informacje na temat podziału pamięci.



Rysunek 19. Dostęp do systemu plików urządzenia mobilnego



Rysunek 20. Wykres kołowy obciążenia procesora w ostatnim czasie

Podgląd logów

Bardzo przydatnym narzędziem, również dla twórców programujących w Cordovie jest LogCat. Pozwala on przeglądać informacje o zdarzeniach zachodzących w urządzeniu i filtrować je na różne sposoby. Filtrowanie ma tu ogromne znaczenie, gdyż liczba zdarzeń, jakie są rejestrowane w każdej sekundzie pracy telefonu jest bardzo duża.

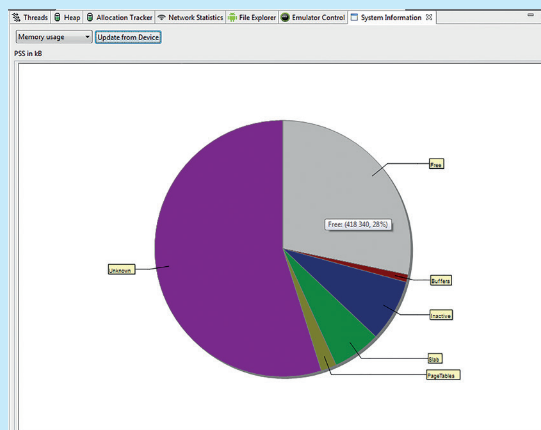
Okno LogCat domyślnie znajduje się na dole ekranu i jest raczej nieduże (**rysunek 22**). Jego główną część zajmuje tabelka z logami. Kolejne kolumny zawierają informacje o:

- poziomie ważności danego wpisu,
- czasie zarejestrowania wpisu,
- identyfikatorze procesu, który dane zdarzenie wywołał,
- identyfikatorze wątku bezpośrednio generującego wpis,
- nazwie aplikacji, która spowodowała uruchomienie danego procesu i wątku,
- tag, pozwalający grupować wpisy na różne sposoby,
- treść logu.

Poszczególnym poziomom ważności odpowiadają różne kolory. Poziomy te to: **assert**, **error**, **warn**, **info**, **debug** i **verbose**. Wybranie ostatniego z nich, a więc najniższego sprawi, że wyświetlane będą komunikaty wszystkich poziomów. Każdy wyższy poziom sprawia, że komunikaty z niższego poziomu nie są pokazywane.

Czytelność logów znacząco utrudnia fakt, że urządzenie, zarówno emulator, jak i prawdziwy telefon, ciągle generują logi nawet na poziomie **error**. Błędy te praktycznie nie zależą od tworzonej aplikacji i należy je pomijać. Problemem jest, że przeszkadzają w znajdowaniu istotnych informacji szczególnie, że bufor rejestrowanych komunikatów nie jest nieskończony i szybko się przepełnia, powodując usunięcie starszych wpisów, gdy pojawiają się nowe. Dlatego warto korzystać z filtrów.

W praktyce, w przypadku deweloperów tworzących aplikacje Cordovy, warto stworzyć sobie filtr, który będzie pokazywał tylko komunikaty pochodzące z programu o interesującej nas nazwie. Posługiwanie się



Rysunek 21. Wykres kołowy aktualnej zajętości pamięci

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hhcxxtt**

numerem procesu lub wątku nie sprawdza się, gdyż nierzadko wywoływane zewnętrzne biblioteki obsługiwane są przez inne wątki i nie wyświetlą się. A to właśnie w nich najczęściej pojawiają się problemy, wymagające skorzystania z narzędzia debugującego.

Filtry można dodawać na żywo, poprzez wpisanie odpowiednich poszukiwanych fragmentów (z ew. prefiksami) w górnej części okna LogCat lub za pomocą lewego fragmentu okna, który pozwala zapisać raz wprowadzony filtr. To chyba najwygodniejsze rozwiązanie (**rysunek 23**). Warto też skorzystać ze znajdującego się na skrajnej, prawej stronie okna LogCat przycisku strzałki, który blokuje ciągłe przewijanie widoku logów.

Dobrym sposobem na wykorzystanie LogCata jest umieszczanie w kodzie własnych poleceń wysyłania komunikatów do konsoli systemowej. Można to robić za pomocą poleceń JavaScriptowych **console.log(), console.error()**, **console.warn()**, **console.info()** i **console.debug()**, co powoduje wysłanie komunikatu o zadanym poziomie ważności. Co więcej domyślnie wysyłane z JavaScriptu w aplikacji Cordovy logi mają swój specyficzny tag, który zależy od wersji systemu, na której są uruchomione. To ułatwia ich odnajdywanie w rejestrze logów.

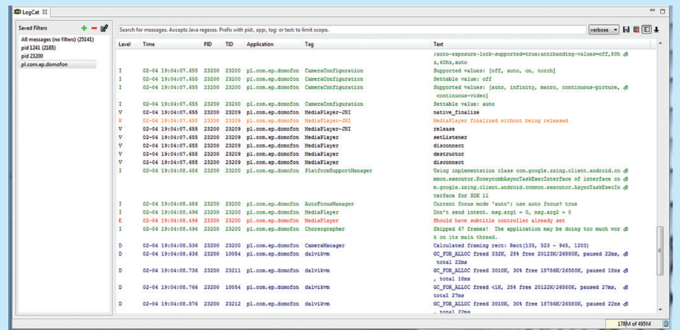
Umiejętne posługiwanie się poleceniami takimi jak **console.log()** i LogCatem pozwoli wykryć momenty, w których coś nie działa tak, jak powinno nawet, gdy wina leży po stronie wtyczki Cordovy. Bardziej doświadczony programista może nawet być w stanie samodzielnie naprawić błąd, bez znajomości języka Java – niekiedy komunikaty błędów z wtyczek wyraźnie wskazują, że problem leży w składni, która przykładowo nie jest już kompatybilna z daną wersją systemu operacyjnego. Wtedy to wystarczy pozmieniać kilka rzeczy w kodzie wtyczki i ponownie skompilować aplikację, usuwając w ten sposób zewnętrzny błąd. Jeśli taki zabieg przekracza umiejętności twórcy aplikacji, może on poszukać alternatywnej wtyczki, która nie będzie podatna na specyficzny przypadek testowy, jaki występuje w danym programie.

Konsola debugera

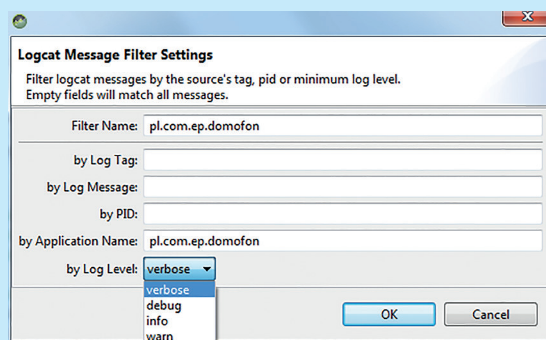
Pozostałe okna Android Device Monitora nie mają większego znaczenia dla programisty tworzącego aplikacje Cordovy. Czasem tylko, gdy pojawi się jakiś problem z działaniem narzędzia, można zajrzeć do okna konsoli systemowej, które zawiera różne komunikaty – najczęściej informacje o błędach dotyczących działania poszczególnych modułów ADM. Jeden z przycisków w oknie konsoli umożliwia przełączanie się pomiędzy informacjami pochodzącymi z poszczególnych modułów, np. z DDMS lub OpenGL Trace View.

Opcje debugowania w telefonie

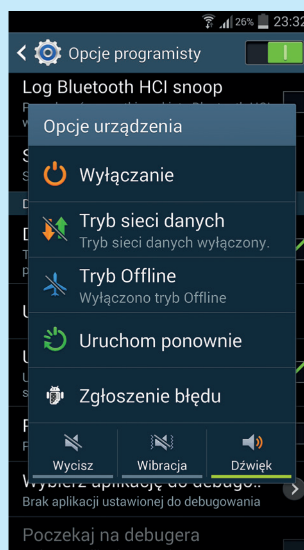
Podczas debugowania dowolnego rodzaju aplikacji androidowej pomocne mogą być możliwości zaszyte w ramach Opcji Programisty, w ustawieniach systemowych urządzenia. Wiele z nich pozwala na diagnozowanie aplikacji bez udziału zewnętrznego debugera. W poprzedniej części kursu obiecaliśmy, że je opiszemy. Ich polskie i angielskie nazwy oraz krótkie opisy zostały podane w **tabeli 2**. Nazwy i zestaw opcji mogą się nieco od siebie różnić dla urządzeń mobilnych



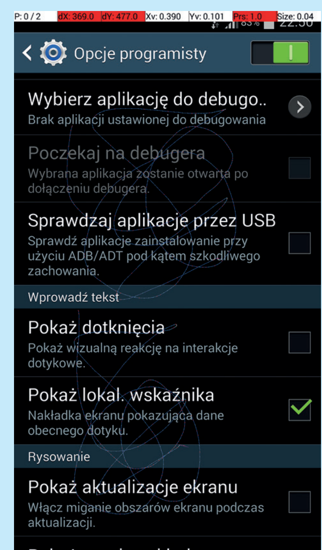
Rysunek 22. Okno narzędzia LogCat



Rysunek 23. Deklarowanie filtru narzędzia LogCat



Rysunek 24. Opcja „Zgłoszenie błędu" dodana dzięki Opcjom Programisty w ustawieniach telefonu



Rysunek 25. Podgląd zarejestrowanego ruchu dotyku na urządzeniu mobilnym. Widać zarejestrowane ścieżki, a na górnym pasku pokazane jest m.in., że w aktualnej chwili 0 z wcześniej wykrytych 2 punktów dotyku jest obecnych.

z innymi wersjami systemów i produkowanych przez różnych producentów.

Z punktu widzenia twórcy aplikacji mobilnej w oparciu o Cordovę, wartościowa będzie tylko część opcji. Jedną z nich jest funkcja rejestrowania pakietów wysyłanych przez interfejs Bluetooth (Log Bluetooth HCI snoop). Przydatne może być też pozorowane położenie, dzięki któremu na prawdziwym telefonie można symulować ruch w przestrzeni, a nie tylko na emulatorze, jak to pokazano w niniejszej części kursu.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hcxxtt

Wydanie specjalne „Raspberry Pi”

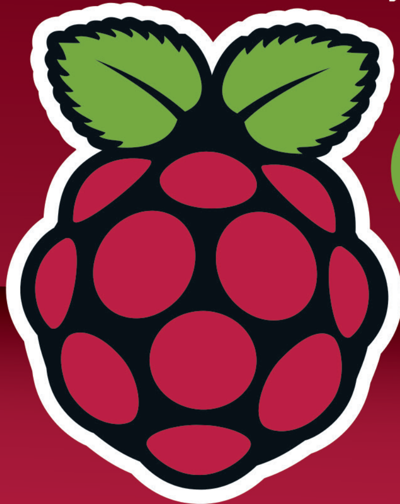
to polski przekład światowego bestsellera na temat słynnego minikomputera

WYDANIE SPECJALNE „MŁODEGO TECHNIKA” NR 1/2015

Raspberry Pi

Ależ to bardzo proste!

Jak w pełni wykorzystać możliwości minikomputera Raspberry Pi



196 pomysłów i porad

KOMPENDIUM DLA NIEELEKTRONIKÓW

ROZPOCZĘCIE PRACY | PODSTAWOWE UMIEJĘTNOŚCI | PROGRAMOWANIE | PROJEKTY

cena: 34 zł (w tym 5% VAT)

To kompendium wiedzy o konfiguracji i sposobach programowania tego uniwersalnego urządzenia oraz prawie dwieście pomysłów i sztuczek aplikacyjnych

Nie będziesz rozczarowany!

Nie musisz być elektronikiem, aby zaprzęgnąć Raspberry Pi do wykonywania niezliczonych rodzajów funkcji i aplikacji

Z tym poradnikiem możesz to osiągnąć!

Szukaj na www.UlubionyKiosk.pl

(przesyłka GRATIS)

Rysunek 26. Podgląd obciążenia procesora różnymi wątkami, wyświetlany na ekranie urządzenia, w trakcie działania aplikacji

Rysunek 27. Podgląd elementów interfejsu, wyświetlany bezpośrednio na urządzeniu mobilnym

Bardzo pomocnymi narzędziami są mechanizmy pokazywania dotknięć, a szczególnie śledzenia wskaźnika (**rysunek 25**). Po włączeniu tej funkcji, oprócz śladów dotknięć na ekranie, na górnym wąskim pasku pojawia się informacja m.in. o aktualnej liczbie wykrywanych punktów, zarejestrowanym przesunięciu, a nawet o rozmiarze wykrywanych punktów dotyku. Ważne są też funkcje z grupy Debugowanie, która pozwala m.in. zautoryzować narzędzia do debugowania czy wybrać aplikację, która ma być poddawana testowaniu i uniemożliwić jej uruchomienie, do momentu włączenia debugera.

Szacowanie wpływu stworzonej przez nas aplikacji, np. w trakcie działania w tle można ocenić korzystając z funkcji wyświetlania zużycia procesora przez poszczególne wątki (**rysunek 26**). Pozwala to łatwo stwierdzić, czy nie zużywamy zbyt wielu zasobów systemowych.

Inną ciekawą i użyteczną, choć nieco mniej przydatną opcją dla twórców aplikacji w Cordovie jest Pokazywanie granic układu, które wskazuje jaką przestrzeń zajmują poszczególne elementy, składające się na całość wyświetlanej treści (**rysunek 27**).

Podsumowanie

Ta i poprzednia część kursu powinny wystarczyć do przygotowania dopracowanej i zoptymalizowanej aplikacji androidowej, pozbawionej błędów. A skoro tak, to gotową aplikacją można się już podzielić – np. poprzez wprowadzenie jej do dystrybucji w ramach któregoś ze sklepów z aplikacjami. Jeśli program ma być oferowany darmowo albo promowany w Internecie, można też rozważyć instalacje modułów do obsługi reklam lub do integracji z serwisami społecznościowymi. W końcu nierzadko warto też wprowadzić obsługę map lub powiadomień typu PUSH, wysyłanych za pośrednictwem twórców danego systemu operacyjnego. Te i inne zagadnienia będą tematem kolejnych części kursu programowania aplikacji mobilnych dla elektroników.

Marcin Karbowniczek, EP

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hcxxtt**