**todo**

1. [todo 1 pośrednie wywołanie intencji kod 21](#_Toc394764893)
2. [todo 2 popraw spisy!!!!!!!!!!!!!!! 37](#_Toc394764894)

**Spis treści**

[1. Wstęp 3](#_Toc394762938)

[1.1. Problematyka i zakres pracy 3](#_Toc394762939)

[1.2. Cele i zakres pracy 4](#_Toc394762940)

[1.3. Przegląd literatury 5](#_Toc394762941)

[1.4. Układ pracy 6](#_Toc394762942)

[2. Podstawowe informacje o Androidzie 8](#_Toc394762943)

[2.1. Historia Androida 8](#_Toc394762946)

[2.2. Android Studio 8](#_Toc394762947)

[2.3. Android Manifest 9](#_Toc394762948)

[2.4. Aktywności 11](#_Toc394762949)

[2.5. Uprawnienia 13](#_Toc394762950)

[2.6. Zasoby aplikacji 14](#_Toc394762951)

[2.7. Serwisy 17](#_Toc394762952)

[3. Intencje oraz BroadcastReceivery 20](#_Toc394762953)

[3.1. Intencje 20](#_Toc394762954)

[3.1.1. Wstęp 20](#_Toc394762955)

[3.1.2. Uruchamianie aktywności i serwisów przy pomocy intencji 20](#_Toc394762956)

[3.2. Natywne akcje Androida 22](#_Toc394762957)

[3.3. Dodawanie dodatkowych informacji do intencji 25](#_Toc394762958)

[3.3.1. Uri 25](#_Toc394762959)

[3.3.2. Mapa dodatkowych danych 25](#_Toc394762960)

[3.3.3. Flagi 26](#_Toc394762961)

[3.4. Podstawowe informacje o IntentFiters 26](#_Toc394762962)

[3.4.1. Budowa filtru intencji 26](#_Toc394762963)

[3.4.2. Jak system rozwiązuje filtry intencji 26](#_Toc394762964)

[3.5. BroadcastReceivery 26](#_Toc394762965)

[3.5.1. Implementacja klasy BroadcastReceiver 27](#_Toc394762966)

[3.5.2. Rejestracja odbiornika 27](#_Toc394762967)

[3.5.3. Wysyłanie wiadomości oraz cykl życia odbiornika 29](#_Toc394762968)

[3.6. Lokalne BroadcastReceivery 29](#_Toc394762969)

[3.7. BroadcastReceivery udostępniane przez Androida 30](#_Toc394762970)

[3.7.1. Wydarzenia dostępne w klasie Intent do powszechnego użytkowania 30](#_Toc394762971)

[3.7.2. Przykłady wydarzenie z innych klas dostępnych w API Androida 32](#_Toc394762972)

[3.8. Użycie BroadcastReceivera oraz intencji na przykładzie zmian stanu modułu Wi-Fi 33](#_Toc394762973)

[Bibliografia 34](#_Toc394762974)

[Spis rysunków 36](#_Toc394762975)

[Spis tabel 37](#_Toc394762976)

[Spis kodów źródłowych 38](#_Toc394762977)

# Wstęp

## Problematyka i zakres pracy

W pierwszym kwartale 2014 roku trzy na pięć sprzedaży telefonów komórkowych nosiło nazwę „inteligentny” [1]. Smartphone[[1]](#footnote-1), jak potocznie   
na urządzenia tego typu się mówi, nie tylko zawładnęły niemałym rynkiem telefonów bezprzewodowych, ale także znacznie poprawiły komfort naszego życia. Od obsługi wiadomości SMS, po zarządzanie pocztą elektroniczną,   
do sterowania temperaturą w naszych mieszkaniach – lista rzeczy, które nie potrafią te urządzenia z każdym dniem zmniejszą jeszcze bardziej.

Nic dziwnego, że tym rynkiem, który jest wart miliardy dolarów rocznie, zainteresowały się wielkie firmy. Swoje rozwiązania w tej dziedzinie technologii   
na przestrzeni ostatnich lat pokazały takie formy jak: Microsoft, Apple czy Google. W swojej pracy zapoznam się jedną z części systemu firmy z Mountain View odpowiedzialną za komunikacji miedzy komponentami zainstalowanymi   
na urządzeniu. Dzięki systemowi nadajników i odbiorników program, które będzie odbierał wiadomości SMS nie musi posiadać dodatkowego wątku odpowiedzialnego za sprawdzenie czy wiadomość tekstowa przyszła, tylko potrzebuje wyczekiwać informacji od części systemu odpowiedzialnej   
za odbieranie krótkich wiadomości. Taki sposób rozwiązania komunikacji między aplikacja w Android przysporzył się do znacznego ograniczenia używanych zasobów przez oprogramowanie działające na tym systemie.

Częściom badawczą mojej pracy było wykonanie aplikacji przedstawiającej działanie tego mechanizmu w Androidzie. Program ten wykorzystuje interfejsy programistyczne, który zostały udostępnione przez firmę Google. Sama aplikacja jest napisana w wersji standardowej języka Java. Niemniej jednak sposób, w jaki został zaprojektowany interfejs programistyczny Androida różni się od tego zaproponowanego przez Oracle. Programiści znający już podstawową edycje Javy zaczynający swoją przygodę z systemem od firmy z Mountain View muszą zapoznać się nie tylko z nazwami udostępnionych klas i metod, ale z sposobem tworzenia interfejsu użytkownika czy komunikacji między systemem a aplikacją. Z tego powodu uważam, że moja praca nie tylko zapoznała czytelnika z systemem rozsyłania i odzierania informacji w Androidzie, ale także pokazała   
go z podstawowymi aspektami tworzenia oprogramowania na ten system.

W części teoretycznej zostaną opisanego takie zagadnienia jak:

* Jak korzystać z powyższych interfejsów
* Komponenty systemu Android, z których można pobierać lub wysyłać informacje potrzebne w działaniu programu
* Klasy poboczne (tj. **IntentFilter**), które są potrzebne w korzystaniu   
  z tych interfejsów
* Zasięg działania obiektów typu **Intent** oraz **BroadcastReceiver**

## Cele i zakres pracy

Głównym celem pracy jest przedstawienie w języku polskim mechanizmu komunikacji między komponentami w systemie Android. Postanowiłem wykorzystać moją prace dyplomową do zaprezentowania tego zagadnienia, ponieważ podczas zaczęcia nauki programowania na ten system natrafiłem   
na dużą, dla sporej grupy ludzi, barierę – barierę językową.

Z powodu przedstawionego w poprzednim akapicie, moja praca skupi się   
na opisanie jednych z najważniejszych mechanizmów w systemie od Google, mianowicie na klasach **Intent** oraz **BroadcastReceiver**. Te obiekty ułatwiają komunikacje miedzy komponentami aplikacji. Trudno wyobrazić sobie scenariusz gdzie, mając na uwadze jak najmniejsze zużycie procesora i pamięci operacyjnej, na jednoczesnym istnieniu dwóch programów na tym samym urządzeniu, które korzystają z osobnych wątków do wykonania **tej samej długiej operacji**.

Pomniejszym celem mojej pracy dyplomowej jest przedstawienie także takich części programu pisanego pod Androida jak:

* Android Manifest
* Aktywności
* Zarządzenie i rodzaje zasobów, z których korzystają aplikacje Androidowe
* Serwisy
* Uprawnienia

Powyższe komponenty są częścią sporej ilości programów mobilnych. Bez znajomości tych technologii trudno, by było zacząć pisać aplikacje Androidowe jak i opisać tytułowy mechanizm.

## Przegląd literatury

Podstawowym źródłem informacji na temat tworzenia aplikacji na Androida jest jego dokumentacja [2]. Wiedza ukazania na tej stronie internetowej jest przejrzysta oraz napisana łatwym do zrozumienia językiem. Stronie nie tylko zawiera opis API[[2]](#footnote-2) Androida, ale także przykłady zastosowań komponentów   
w niej zawartych [3]. O ile dokumentacja dostarczona przez Google jest bardzo dobrze napisana w większości obszarów, to tak niektóre nie zostały w ogóle wspomniane. Przykładem komponentu, który nie został opisany w tutorialach[[3]](#footnote-3), jest klasa **BroadcastReceiver**. Na szczęście to zagadnie zostało opisane przez osoby spoza firmy z Mountain View [4].

O ile wiedza na temat tworzenia aplikacji działający na Androidzie dostępną w Internecie jest duża i najczęściej darmowa, ale za to jest ona rozsiana po całej sieci. Czas spędzany na przeszukiwaniu Internetu w poszukiwaniu dobrej jakości artykułu na temat nas interesujący jest czasami ogromny, dlatego istnienie książek z tej domeny nauki jest bardzo potrzebny. Jednym z najlepszych podręczników mówiących o programowaniu na Androida mim zdaniem należy wolumen autorstwa Reto Meier [5]. Książka ta to jeden z najlepszych kompendiów wiedzy zawartej jednym podręcznym opakowaniu. Zawarte w min jest wszystko, co musi wiedzieć każdy, kto chce programować na system od Google.

W języku polskim trudno o dobry podręcznik traktujący o tworzeniu aplikacji na Androida. Najbliżej mianu „kompendium wiedzy” jest przetłumaczona książka autorstwa Shane Conder oraz Lauren Darcey [5]. Podręcznik ten zawiera informacje potrzebne do bez problemowego napisania pierwszej poważnej aplikacji na Androida.

## Układ pracy

Tematem pracy jest mechanizm rozsyłania informacji o zdarzeniach   
w systemie Android, a głównym cele mojej pracy jest przedstawienie tego problemu w języku polskim. W pierwszym rozdziale zawiera wstęp i cele pracy dyplomowej.

Drugi rozdział to przedstawienie podstawowych informacji potrzebnych   
do opisu tytułowego mechanizmu. Ta część pracy zawiera między innymi wiedzę na temat:

* Manifestu aplikacji Androidowej
* Zasobów, które używa aplikacja pod Androida
* Uprawnień, które może uzyskać program

W trzecim rozdziale został opisany tytułowy mechanizm. Informacje w nim zawarte dotyczą:

* Tworzenia, używania obiektów klas **Intent** oraz **BroadcastRecaiver**
* Opisu identyfikacji **BroadcastReceivera** przez system.
* Zasięgu nadajnika oraz odbiornika
* Natywnych akcji Androida

Czwarty rozdział to opis programu napisanego na potrzeby pracy. Zostało tan opisane architektura aplikacji oraz przykładowe jej zastosowanie.

Ostatnim rozdziałem jest podsumowanie, w którym zaprezentowano rezultaty pracy. Wynika z nich przede wszystkim, że bez mechanizmu **Intentów** oraz **BroadcastReveiverów** aplikacje pisane pod Androida wymagałyby urządzeń   
z dużo ilością pamięci operacyjnej oraz szybkim procesorem. Głównym rezultatem pracy jest ukazanie wad i zalet mechanizmu rozsyłania informacji   
o zdarzaniach w systemie od firmy Google oraz jego opis w języku polskim.

# Podstawowe informacje o Androidzie



## Historia Androida

W roku 2008 na Mobile World Congress w Barcelonie firma Google pokazała pierwsze prototypowe urządzenie z ich nowym systemem mobilnym   
na pokładzie nazwanym Android [7]. Był to telefon przypominający popularne   
w tamtym okresie produkty spółki Research In Motion (teraz BlackBerry). Kilka miesięcy później firma HTC wypuściła do sprzedaży pierwszy telefon w tym systemem [8] – HTC Dream, znany bardziej, jako T-Mobile G1.

Od tego czasu Androida działa na ponad 1 miliardzie urządzeń [9], miał kilkanaście dużych wydań oraz jest obecny na prawie, co drugim sprzedanym smartphone ‘e [10].

Sukces Androida może przypisać modelowi biznesowemu, jaki przyjęła firma z Mountain View:

* System ten jest dostępny, jako projekt open-source, czyli każdy kto tylko ma wiedzie i możliwości może pobrać jego kod i stworzyć własną wersje Androida
* Może działać na prawie każdym urządzeniu wyposażonym   
  w mikroprocesor, na przykład telefony komórkowe, telewizory,   
  a nawet lodówki
* Tworzenie aplikacji na niego jest całkowicie darmowe. Każdy może wejść na stronę dla developerów Androida i pobrać narzędzia potrzebne do tego

## Android Studio

Android Studio to oficjalne **IDE**[[4]](#footnote-4) do tworzenia programów działających pod kontrolą Androida. Składa się z dwóch części:

* **SDK[[5]](#footnote-5)** dla systemu Android\
* Intelij IDEA Community Edition, zmodyfikowany przez firmę Google na potrzeby programowanie na ich system. Przykładowy zrzut ekranu przestawiony te program widać na Rysunku 1.

Rysunek Przykładowy zrzut ekranu z Intelij Android Studio

## Android Manifest

Android Manifest to plik XML, który zawiera opis atrybutów aplikacji.   
W nim są zawarte informacje takie jak: nazwa aplikacji, uprawnienia jakie posiada aplikacja czy **BroadcastRecaivery**. Tabela 1 przedstawia przykładowe znaczniki, które są dostępne oraz ich krótki opis.

Tabela Przykładowe znaczniki, które są dostępne w Android Manifest

|  |  |
| --- | --- |
| Nazwa znaczka | Opis |
| <action> | Zawiera nazwę akcji, przy której będzie uruchamiany **BroadcastReceiver**. Używany w <intent-filter> |
| <activity> | Deklaruje aktywności dostępne w programie. Jeżeli aktywność nie zostanie opisana przez ten znacznik nie będzie widoczna dla systemu i nie zostanie wyświetlona. |
| <application> | Definiuje atrybuty aplikacji takie jak: jej nazwa, ikonę, czy może być debatowana, aktywności używane przez aplikacje, serwisy. |
| <intent-filter> | Może być zdefiniowany wewnątrz znaczników *<activity>*, *<service>* oraz *<reciver>*. Odpowiada  za zdefiniowanie akcji, na które wcześniej wymienione komponenty będą mogły odpowiadać. |
| <manifest> | Główny element w pliku XML’owym. |
| <receiver> | Jeden z dwóch sposobów na zdefiniowanie obiektu typu **BroadcastReceiver**. |
| <service> | Używany go deklaracji obiektów klasy **Service**. Tak jak w przypadku *<activity>*, jeżeli serwis nieopisany tutaj, system nie będzie w wiedział o jego istnieniu. |
| <uses-permission> | Definiuje uprawnienia, które muszą być udzielone aplikacji, by mogła działać poprawnie. |

Powyższy listing przedstawia przykładowy Android Manifest.

|  |
| --- |
| 1 <?xml version=**”1.0”** encoding=**”utf-8”**?>  2 <manifest xmlns:android=**”**[**http://schemas.android.com/apk/res/android**](http://schemas.android.com/apk/res/android)**”**  3 package=**”org.karolgurecki.autotask”**  4 android:versionCode=**”1”**  5 android:versionName=**”1.0”**>  6 <uses-sdk android:minSdkVersion=**”16”**/>  7 <application android:label=**”@string/app\_name”**  8  android:theme=**”@android:style/Theme.DeviceDefault”**>  9 <activity android:name=**”.ui.activities.Main”**  10 android:label=**”@string/app\_name”**>  11<intent-filter>  12<action android:name=**”android.intent.action.MAIN”**/>  13<category android:name=   **”android.intent.category.LAUNCHER”**/>  14</intent-filter>  15</activity>  16<activity android:name=**”.ui.activities.NewTaskActivity”**  17android:label=**”@string/newTaskButtonText”**/>  18<activity android:name=**”.ui.activities.TaskListActivity”**  19android:label=**”@string/taskListButtonText”**/>  20<activity android:name=**”.ui.activities.TaskList”**  21android:label=**”Task List”**/>  22<service android:name=**”.service.StartUpService”**  23android:label=**”Autotask Service”**/>  24<receiver android:name=**”org.karolgurecki.autotask.**  25 **service.StartUpBroadcastReceiver”**>  26<intent-filter>  27<action android:name=**”android.intent.action.**  **ACTION\_EXTERNAL\_APPLICATIONS\_AVAILABLE”**/>  28<action android:name=**”android.intent.action.**  **BOOT\_COMPLETED”**/>  29</intent-filter>  30</receiver>  31</application>  32<uses-permission android:name=  “**android.permission.READ\_EXTERNAL\_STORAGE”**/>  33<uses-permission android:name=  “**android.permission.WRITE\_EXTERNAL\_STORAGE”**/>  34<uses-permission android:name=  “**android.permission.RECEIVE\_BOOT\_COMPLETED”**/>  35<uses-permission android:name=**”android.permission.BLUETOOTH”**/>  36</manifest>  Listing Przykładowy plik AndrodManifest.xml |

## Aktywności

Za interfejs użytkownika aplikacji w Androidzie odpowiadają obiekty klasy **Activity**. Większość programów posiada wiele luźno ze sobą powiązanych aktywności.

Zwykle aplikacje mają jeden główny obiekt **Activity**, który jest uruchamiany zaraz po jej włączeniu. Aby system wiedział, że dana aktywność jest „główna” musi posiadać **IntentFilter**, który zawiera „android.intent.action.MAIN” jako akcje oraz kategorie „android.intent.category.LAUNCHER”   
– przykład takiego zastosowania widać na listingu pierwszych w liniaka 12 i 13.

Powyższy sposób działa jedynie, gdy chcemy, aby jakaś aktywność była uruchomiona jako pierwsza. Aby wyświetlić inne Activity, na przykład   
po naciśnięciu przycisku, trzeba utworzyć **Intent** używając obiektu **Class** aktywności, którą chcemy pokazać. Listing 2 przedstawia przykład uruchomienia aktywności z kodu programu.

|  |
| --- |
| 1 Intent intent **=** **new** Intent**(**Main**.this,**TaskListActivity**.**class**);**  2 startActivity**(**intent**);** |

Listing Tworzenie Activity przy pomocy kodu

Każde **Activity** w Androidzie ma kilka możliwych stanów, w których może się znaleźć. Sekwencje przechodzenia miedzy nimi widać na rysunku 1.



Rysunek Cykl życia aktywności. Źródło: [11]

Podczas tworzenia nowego klasy, które rozszerza **Activity** musimy napisać własną implementacje metodyonCreate(). Reszta stanów aktywności posiad swoje domyślne definicje, ale nic nie stoi na przeszkodzie, aby programista napisał własne.

Trzeba także wspomnieć, że system posiada tak znany **Back Stack**, czyli stos aktywności, które zostały uruchomiono przez użytkownika. Ta kolekcja jest kolejkowana przy pomocy algorytmu **LIFO[[6]](#footnote-6)**, czyli ostatnia aktywność na stosie jest aktualnie wyświetlana przez system.

## Uprawnienia

Z powodu bezpieczeństwa w Androidzie zastosowaniom system uprawnień (ang. *Premissions*). Domyślnie aplikacja ma podstawowe przywileje – może jedynie zarządzać swoimi danymi. Programista wiedząc, że będzie potrzebować dostępu na przykład do stanu modułu Bluetooth, musi powiadomić o tym fakcie system. Odbywa się to przy pomocy dodania znacznika *<use-permission>*   
w AndroidManifest.xml. Android posiada szereg wbudowanych uprawnień, których przykłady widzimy w tabeli 2.

Tabela Przykładowe uprawnienie, które udostępnia Android

|  |  |
| --- | --- |
| Nazwa przywileju | Opis |
| android.permission.  ACCESS\_WIFI\_STATE | Umożliwia aplikacji dostęp  do informacji na temat sieci Wi-Fi |
| android.permission.  BATTERY\_STATS | Pozwala na odczyt stanu baterii. |
| android.permission.  BLUETOOTH\_ADMIN | Umożliwia odnajdywanie urządzeń przez protokół Bluetooth oraz łączenie z nimi. |
| Android.Permission.INTERNET | Potrzebne, aby program mógł korzystać z zasobów dostępnych w Internecie. |
| android.permission.VIBRATE | Umożliwia do funkcji wibracji  w urządzeniu. |
| android.permission.RECEIVE\_SMS | Pozwala na odczyt wiadomości SMS, które przychodzą do urządzenia. |

API Androida pozwala także na definiowane własnych uprawnień przez programistę. Pozwała ono do zablokowania dostępu do takich elementów aplikacji jak: serwisy, aktywności czy broadcastReceivery przez inne aplikacje. Aby tego dokonać programista musi:

* Dodać znacznik *<permission>* w AndroidManifest.xml z definicją upoważnienia
* Deklaracji elementu, który chce zabezpieczyć musi atrybutowi *android:permission* nadać wartość taką samą jak nazwa przywileju, który wcześniej zdefiniował

## Zasoby aplikacji

Oddzielenie zasobów, które używa aplikacji od jej kodu, jest uważane przez firmę Google, jako jeden z **best practice[[7]](#footnote-7)** przy programowaniu na ich system.   
Z tego powodu Android został wyposażony w szereg udogodnień dla programistów. Te proces pozwala w prosty sposób przystosować program między innymi do różnych języków czy wielkości ekranu.

Android wymaga, aby wszystkie zasoby były umieszczone w katalogu res/, który znajduję się w głównym folderze projektu. Tutaj trzeba także nadmienić,   
że różne typy zasobów muszą być umieszczone w różnych, określonych przez specyfikacje, pod folderach.

Dokumentacja dostarczona przez Google’a dzieli zasoby na dwa podstawowe rodzaje:

* Alternatywne, czyli zasoby, które są tworzone z myślą o określonej konfiguracji (na przykład plik z *layoutem* dla danej wielkości ekranu). System rozpoznaje, że dana grupa zasobów odnosi się do określonej konfiguracji, poprzez odpowiedni modyfikator w nazwie pod folderu, w którym się ona znajduje (na przykład, jeżeli katalog się nazywa values-pl to Android wiem, iż w nim są pliki odnoszące się do lokalizacji aplikacji w języku polskim).
* Domyślne – to zasoby, które system zacznie używać, jeżeli nie znajdzie żadnych konfiguracji pod aktualną specyfikacje.

Każdy zasób może być użyty w kodzie aplikacji. Odbywa się to przy pomocy podklas klasy R. Klasa ta jest automatycznie generowana przez kompilator podczas kompilacji projektu. Kompilator czyta zawartość folderu res/ wraz jego pod katalogów i umieszcza zserializowane znajdujących się w nich plików, w odpowiednich podklasach klasy R.

Dodatkowo powyższe rodzaje zasobów dzielą się na kilka typów:

* Animacje – zawiera informacje na temat animacji zdefiniowanych przez autora aplikacji. Poszczególne klatki powinny być umieszczone w katalogu res/drawable/ oraz używane z Javy przy pomocy klasy R.drawable. Natomiast przejścia między poszczególnymi klatkami powinny się znaleźć w res/anim/, a korzystać można z nich przy pomocy R.anim.
* Listy stanów kolorów – definiuje informacje na temat kolorów używanych w widokach. Dane te są zapisane w /res/color/ oraz dostępne przy pomocy R.color.
* Grafiki – definiuje informacje na temat danych graficznych w postaci bitmap lub plików XML. Powinny być umieszczone w katalogu /res/drawable/, a używane z poziomu kodu przy pomocy R.drawable.
* Layout – definicje układ elementów na poszczególnych widokach. Pliki XML (przykład widzimy na listingu 3) z tymi informacjami umieszczone muszą być w /res/layout/, a dostęp do nich zapewnia R.layout.

|  |
| --- |
| 1<?xml version=**"1.0"** encoding=**"utf-8"**?>  2<LinearLayout xmlns:android= 3 **"http://schemas.android.com/apk/res/android"**  4 android:orientation=**"vertical"**  5 android:layout\_width=**"fill\_parent"**  6 android:layout\_height=**"fill\_parent"**>  7<Button android:layout\_width=**"match\_parent"**  8 android:layout\_height=**"fill\_parent"**  9 android:text=**"@string/newTaskButtonText"**  10 android:id=**"@+id/newTaskbutton"**  11 android:layout\_gravity=**"left|center\_vertical"**  12 android:layout\_weight=**"0.5"**/>  13<Button android:layout\_width=**"match\_parent"**  14 android:layout\_height=**"fill\_parent"**  15 android:text=**"@string/taskListButtonText"**  16 android:id=**"@+id/taskListButton"**  17 android:layout\_gravity=**"left|center\_vertical"**  18 android:layout\_weight=**"0.5"**/>  19</LinearLayout> |

Listing Przykładowy plik XML zawierający informacje na temat układu widoku

* Menu – zawiera informacje o zawartości menu w aplikacji. Z kodu dostępne poprzez R.menu, a pliki znajdują się w /res/menu/.
* Style – definiują wygląd interfejs użytkownika. Pliki należy umieszczać w /res/style, a dostęp zapewnia R.style.
* Inne zasoby – zasoby takie jak stringi[[8]](#footnote-8), liczby całkowite, tablice muszą być umieszczone w /res/values/. Każdy typ ma swoją podklasę (na przykład R.string, R.integer). Listing 4 przedstawia przykładowy plik strings.xml.

|  |
| --- |
| 1 <?xml version="1.0" encoding="utf-8"?>  2 <resources>  3 <string name="app\_name">AutoTask</string>  4 <string name="newTaskButtonText">New Task</string>  5 <string name="taskListButtonText">Task List</string>  6 <string name="newTaskNameEditText">Task name</string>  7 <string name="actions">Actions</string>  8 <string name="triggers">Triggers</string>  9 <string name="addNew">Add new...</string>  10 <string name="add\_trigger">Add trigger</string>  11 <string name="add\_action">Add action</string>  12 <string name="finish">Finish</string>  13 <string name="saveTaskTitleAlert">Save task</string>  14 <string name="doYouWantSave">Do you want to save this   task?</string>  15 <string name="yes">Yes</string>  16 <string name="no">No</string>  17 <string name="cancel">Cancel</string>  18 <string name="autotask">Autotask</string>  19 <string name="off">Off</string>  20 <string name="on">On</string>  21 <string name="confirm">Confirm</string>  22 <string name="bluetoothTriggerConfigTitle">Bluetooth trigger   configuration</string>  23 <string name="bluetoothTrigger">Bluetooth trigger</string>  24 <string name="ifString">If</string>  25 <string name="toastDialogTitle">Toast Text   Confuguration</string>  26 <string name="display">Display</string>  27 <string name="toastActionName">Toast Action</string>  28 </resources> |
|  |

Listing Przykładowy plik strings.xml

Google w API Androida umieściło także klasę android.R zawierającą zasoby, które często są wykorzystywane przez programistów. Klasa   
ta ma w sobie referencje do takich obiektów jak: domyślne style, ikony, czy często używane stringi.

## Serwisy

Aktywności w Androidzie są aktywne tylko jak są wyświetlane na ekranie urządzenia. Każda akcja, która trwa długo powoduje, że widok, który obserwuje użytkownik nie odpowiada. Dlatego osoby, które projektowały ten system, postanowiły wprowadzić mechanizmy do przetwarzania informacji w tle.   
Po za domyślnymi mechanizmami, które udostępnia Java oddano kilka, które zostały stworzone na potrzeby systemu od Google’a. Jednym z ich przedstawicieli są serwisy.

W odróżnieniu od aktywności – serwisy zostały stworzone z myślą o długo trwałej pracy. Posiadają wyższy priorytet niż widoki, które zostały zamrożone, przez co zostaną usunięte z pamięci tylko, jeżeli systemowi (po uprzednim opróżnieniu stosu uśpionych aktywności) zabraknie pamięci operacyjnej   
do na przykład otworzenia kolejnego widoku. Niemniej Android pozwala   
na skonfigurowanie systemu, aby ten mógł ponownie uruchomić nasz serwis, jeżeli ilość wolnego RAMu na to pozwala.

Aby utworzyć własny serwis programista musi rozszerzyć klasę **Service**.   
W celu odpowiedzenia na komendę uruchomienia komponentu trzeba nadpisać jedną w dwóch metod onStartCommand lub onStart. Z powodu, że ta druga funkcja została oznaczona jako przestarzała w chwili wydania Androida w wersji 2.0, Google zaleca używanie onStartCommand.

|  |
| --- |
| 1 public class StartUpService **extends** Service **{**  2 public static final List**<**String**>** TASK\_PROPERTIES\_NAME\_LIST  3 **=** **new** LinkedList**<>();**  4 private final static FileFilter FILE\_FILTER **=**  5 **new** FileFilter**()** **{**  6 @Override  7 public boolean accept**(**File pathname**)** **{**  8 String name **=** pathname**.**getName**().**toLowerCase**();**  9 **return** name**.**endsWith**(**  10 ConstanceFieldHolder**.** 11 PROPERTIES\_FILE\_EXTENTION**.**toLowerCase**())**  12 **&&** **!**TASK\_PROPERTIES\_NAME\_LIST**.**contains**(**name**);**  13 **}**  14 **};**  15 public static TaskHolderMap TASK\_HOLDER\_MAP**;**  16  17 @Override  18 public void onCreate**()** **{**  19 **super.**onCreate**();**  20 **}**  21  22 @Override  23 public int onStartCommand**(**Intent intent**,** int flags**,** int startId**){**  24 **super.**onStartCommand**(**intent**,** flags**,** startId**);**  25 **if** **(**TASK\_HOLDER\_MAP **==** **null)** **{**  26 TASK\_HOLDER\_MAP **=** **new** TaskHolderMap**();**  27 registerReceiver**(**TASK\_HOLDER\_MAP**,** **new**  28 IntentFilter**(**TaskHolderMap**.**TASK\_HOLDER\_MAP\_ACTION**));**  29 **}**  30 File autoTaskFolder **=** getFilesDir**();**  31 **if** **(**autoTaskFolder**.**exists**())** **{**  32 File**[]** fileArray **=**  33 autoTaskFolder**.**listFiles**(**FILE\_FILTER**);**  34 **for** **(**File file **:** fileArray**)** **{**  35 // ...  36 **}**  37 **}** **else** **if** **(!**autoTaskFolder**.**mkdirs**())** **{**  38 Log**.**e**(**ConstanceFieldHolder**.**AUTOTASK\_TAG**,**  39 "Can't create a AutoTask folder"**);**  40 **}**  41 **return** Service**.**START\_STICKY**;**  42 **}**  43  44 public IBinder onBind**(**Intent intent**)** **{**  45 **return** **null;**  46 **}**  47  48 @Override  49 public boolean stopService**(**Intent name**)** **{**  50 TASK\_HOLDER\_MAP**.**destroyAll**();**  51 unregisterReceiver**(**TASK\_HOLDER\_MAP**);**  52 **return** **super.**stopService**(**name**);**  53 **}**  54**}** |

Listing Fragment StartUpService.java

Na powyższym listingu widzimy fragment serwisu, będącego częścią praktycznej części pracy dyplomowej. Ta implementacja klasy **service** zajmuje się odczytywaniem folderu lokalnego aplikacji w poszukiwaniu plików .properties, które zawierają specyfikacje zdań wykonywanych przez nią (więcej na ten temat zawiera rozdział 4 pracy).

Na koniec trzeba wspomnieć, że tak jak w przypadku aktywności, wszystkie używanie serwisy muszą być zdefiniowane w manifeście aplikacji. Definicje StartUpService można zobaczyć na poniższym listingu.

|  |
| --- |
| 1 <service android:name=".service.StartUpService"  2 android:label="Autotask Service"/> |

Listing 6 Definicja StartUpService w AndroidManifest.xml

# Intencje oraz BroadcastReceivery

## Intencje

### Wstęp

Intencje w Androidzie służą jako mechanizm komunikacyjny miedzy aplikacjami jak i ich komponentami. Stosuje się je do:

* Uruchamiania aktywności oraz serwisów.
* Nadawania informacji o zdarzeniach, które się pojawiły   
  na urządzenie

Są uważane, jako jeden z wielu **best practice** przy programowaniu   
na Androida. Google zaleca używane ich nie tylko przy komunikacji system   
– aplikacja, a także w jej wnętrzu. Pozwala ją na bez problemową modyfikacje jednego komponentu programu bez zmiany referencji do niego.

Android powstał jako system, który można łatwo dostosować do własnych potrzeb. Jednym z owoców tego postanowienia był właśnie mechanizm intencji. Używając ich w łatwy sposób można napisać aplikacje, którą nadpisuje funkcyjności już w nim dostępną dzięki oczekiwaniu na te same akcje,   
co programy systemowe.

Dzięki intencją system jest w stanie z na pozór niepołączonych ze sobą komponentów z różnych aplikacji (niepisanych przez jednego autora) – stworzyć jedną połączoną systemem nadajników i odbiorników całość.

### Uruchamianie aktywności i serwisów przy pomocy intencji

Jak zostało wspomniane w poprzednim podrozdziale intencje między innymi służą do tworzenia nowych aktywności oraz serwisów podczas działania programu. Mamy dwie możliwości ich uruchamiania:

* Bezpośrednio (z ang. *explicitly*), przy pomocy class
* Pośrednio (z ang. *implicitly*), przy użyciu akcje z odpowiednimi danymi (te sposób może być użyty gdzie nie znamy nazwy widoku, który chcemy wyświetlić)

Pierwszy z wymienionych wcześniej sposobów jest najczęściej używany przy startowaniu aktywności lub serwisu, która znajduje się w tej samym programie. System przy wykorzystania obiektu class tego elementu tworzy go, uruchamia   
i przenosi go nas początek stosu widoków. Nowa aktywność jest obsługiwana przez ten sam wątek, który ją otwierał. Natomiast nowo uruchomiony serwis jest tworzony w tym samym wątku co aplikacja, która go wywołuje, ale po jego poprawnym uruchomieniu działa on w oddzielnym wątku. Poniższy listing zawiera przykładowe jednoznaczne wystartowanie aktywności.

|  |
| --- |
| 1 Intent intent **=** **new** Intent**(**Main**.this,**TaskListActivity**.**class**);**  2 startActivity**(**intent**);** |

Listing Przykładowe jednoznaczne uruchomienie aktywności

Innym podejściem do stratowania aktywności jest użycie akcji podczas tworzenia intencji. Dzięki temu mechanizmowi może programista powiedzieć systemowi, że potrzebuje uruchomić aplikacje, która obsłuży określoną czynność, na przykład otworzy daną stronę WWW, co można zaobserwować na listingu 8.

|  |
| --- |
| todo pośrednie wywołanie intencji kod |

Listing Wyświetlenie strony WWW w domyślnej przeglądarce

Na powyższym listingu można zauważyć, że nie podajemy dokładnej informacji na temat widoku, który chcemy wyświetlić. System po przeparsowaniu zmiennej typu **URI**, którą podajemy jako jeden z paramentów przy tworzeniu intencji wie, iż chcemy wyświetlić treść pod podanym adresem WWW. Z tego powodu uruchomi domyślną przeglądarkę internetową lub okno wyboru programu, który ma być odpowiedzialny za wyświetlanie tego typu treści. Przykładowe okno z wyborem aplikacji jest ukazane na rysunku 3.



Rysunek Okno wyboru domyślnej przeglądarki w systemie

## Natywne akcje Androida

Poniższa lista zawiera większość intencji, które udostępnia klasa **Intent**   
(w postaci statycznego stringa) z API Androida:

* **ACTION\_ALL\_APPS** – wyświetla wszystkie zainstalowane aplikacje. Najczęściej używane przez programu typu *Launcher*.
* **ACTION\_ANSWER** – odpowiada za wyświetlenie widoku nadchodzącej rozmowy
* **ACTION\_APP\_ERROR** – ta intencja jest dostarczana to aplikacji, która zainstalowała daną paczkę, po naciśnięciu przycisku „Report”   
  z dialogu pozwalającego zaraportowanie błędu w aplikacji.
* **ACTION\_ASSIST** – wyświetla aplikacje typu **Asystent**   
  po naciśnięciu i przytrzymaniu przycisku domowego.
* **ACTION\_BUG\_REPORT** – wyświetla widok odpowiedzialny raportowanie błędu w aplikacji.
* **ACTION\_CALL** – pozwala na wykonanie połączenia z określonym numerem telefonu.
* **ACTION\_CALL\_BUTTON** – umożliwia pokazanie aplikacji pozwalającej na wybranie numeru telefonicznego.
* **ACTION\_CHOOSER** – Najczęściej używany, jako zamiennik   
  do standardowego widoku wybierania aktywności, którą ma zostać wyświetlona, jeżeli system wiele aplikacji z danej kategorii. Programista może nadać własny tytuł temu oknu dialogowemu, ale   
  za to użytkownik nie ma możliwości ustawienia wybranego programu jako domyślnej aplikacji dla tej akcji.
* **ACTION\_CREATE\_SHORTCUT** – tworzy skrót. Zwraca intencje, którą musi zawierać informacje o
  + Nazwie skrótu
  + Intencji, którą ma uruchamiać
  + Ikonie, jaką ma on posiadać

lub obiekt typu Intent.ShortcutIconResource

* **ACTION\_DELETE** – przeprowadza operacje usunięcia   
  na przekazanych danych.
* **ACTION\_DIAL** – działa podobnie do **ACTION\_CALL**, czyli wyświetla element interfejsu użytkownika odpowiedzialny   
  za wybieranie numeru telefonu. Programista ma możliwość podania numeru telefonu, na który ma zostać wykonane połączenie.
* **ACTION\_EDIT** – pozwala na edycje wygranego zasobu.
* **ACTION\_GET\_CONTENT** – pozwala na poproszenie użytkownika o wybranie elementu wyszczególnionego typu danych (na przykład kontaktu) oraz przekazanie go z powrotem do aplikacji.
* **ACTION\_INSERT** – tworzy nowy element w przekazanej przez programistę ścieżce (na przykład plik, obraz, kontakt).
* **ACTION\_INSERT\_OR\_EDIT** – pozwala na stworzenie nowego elementu lub edycje istniejącego typu danych.
* **ACTION\_MAIN** – jedna z najważniejszych akcji opisanych tutaj. Jest wykorzystywana w większości aplikacji na Androida. Przy jej pomocy definiuje się główny widok programu.
* **ACTION\_OPEN\_DOCUMENT** - pozwala za otworzenie jednego lub wielu dokumentów używając zainstalowanych na urządzeniu **DocumentProvider.**
* **ACTION\_PASTE** – przy użyciu danych znajdujących się w schowku podręcznym, tworzy nowy element w wybranym kontenerze.
* **ACTION\_PICK** – zwraca ścieżkę do wybranego elementu.
* **ACTION\_POWER\_USAGE\_SUMMARY** – wyświetla widok odpowiedzialny za podanie sumarycznych informacji o zużyciu energii.
* **ACTION\_SEACH** – uruchamia widok odpowiedzialny  
  za wyszukiwanie informacji.
* **ACTION\_SEND** – wysyła otrzymane dane przy pomocy **EXTRA\_TEXT** lub **EXTRA\_STREAM** do wybranej przez użytkownika aplikacji z danej kategorii.
* **ACTION\_SEND\_MULTIPLE** – ta intencji w odróżnieniu  
  od **ACTION\_SEND** pozwala wysłać więcej niż jeden element przy jednorazowym wywołaniu.
* **ACTION\_SET\_WALLPAPER** – wyświetla widoku, który pozwala na zmianę tła ekranu blokady lub/i startowego
* **ACTION\_WEB\_SEARCH** – w odróżnieniu od **ACTION\_SEARCH** wyszukuje informacje dostępne tylko Internecie.

Na koniec tego podrozdziału trzeba wspomnieć, że klasa **Intent** nie jest jednym miejscem w API Androida gdzie możemy akcje. Inne klasy gdzie mamy je dostępne to między innymi **Settings**, gdzie są zawarte intencje do widoków  
od wszelakich ustawień na urządzeniu.

## Dodawanie dodatkowych informacji do intencji

Jak w poprzednim podrozdziale zostało wspomniane, iż wiele intencji dostępnych w klasie **Intent** podczas tworzenia, prócz podania ich nazwy komponentu czy akcji, posiada także możliwość dodania dodatkowych danych potrzebnych podczas startu albo działania widoku lub BroadcastReceivera. Można   
to wykonać przy pomocy kilku sposobów – przy pomocy obiektu klasy **Uri**, używając mapy zwanej **EXTRA** lub korzystając z flag.

### Uri

Pierwszy typ danych, z którego można korzystać przy tworzeniu intencji jest obiekt klasy **Uri**. Jak sama nazwa klasy wskazuje[[9]](#footnote-9) jej obiekty nie zawierają samej informacji tylko odnośnik do niej. W trakcie konstrukcji elementu tego typu trzeba podać stringa, który zawiera informacje o typie oraz miejscu ich przechowywania. Danych w postaci obiektu **Uri** stosuje się na przykład jak chcemy wyświetlić kontakt przy pomocy ACTION\_VIEW o nazwie *mama*, taki odnośnik wygląda następująco – **content://contacts/people/mama**.

Warto tutaj zaznaczyć, że jeżeli programista chce wykorzystać ten typ danych podczas tworzenia intencji musi skorzystać z odpowiedniego konstruktora lub jedno z trzech udostępnionych do tego setterów[[10]](#footnote-10). Jeżeli zostanie wybrany dwóch sposób oraz zajdzie potrzeba ustawienia jednocześnie typu danych oraz ścieżki do elementu, powinna być użyta funkcja **setDataAndType()**. Jest   
to spowodowane faktem, iż metody nadające wartość tych zmiennych w klasie **Intent** wzajemnie się wykluczają.

### Mapa dodatkowych danych

Innym sposobem dodawania informacji do intencji dodawanie ich do mapy reprezentowanej przez obiekt klasy **Bundle**. Zmienna ta może przechowywać element każdego typu łącznie kluczem, który jest stringiem. Klasa **Bundle** jak   
i **Intent** posiada szereg metod, które pozwalają zapisać oraz odczytać informacje w nich zawarte. Jednym z **best practice** stosowanym przy tym podejściu jest zapis kluczy jako statycznych publicznych zmienny stringowych w celu ułatwienia korzystania z nich przez inne komponenty.

### Flagi

Dzięki flagą w intencji Android może określić jak ma się zachować podczas uruchamiania oraz po zamknięciu widoku. Po przez ten mechanizm programista może powiedzieć systemowi, między innymi, aby ten uruchomił aktywność w tle lub nie zostawił informacji o niej w liście ostatnio uruchamianych aplikacji.

## Podstawowe informacje o IntentFiters

### Budowa filtru intencji

### Jak system rozwiązuje filtry intencji

## BroadcastReceivery

Jak zostało napisane w początkowym rozdziale pracy, osoby odpowiedzialne   
za stworzenie Androida wiedząc, że ich system będzie działał na urządzeniach   
z ograniczoną mocną obliczeniową oraz działające na baterii, chcieli w jak największym stopniu ograniczyć wielokrotne przetwarzanie tych samych danych przez różne komponenty w nim zainstalowane. Dzieckiem tego podejścia jest mechanizm komunikacji między zainstalowanymi aplikacjami na urządzeniu. Działa on na zasadzie sieci nadajników oraz odbiorników. BroadcastReceivery pełnią w tym systemie tą drugą rolę

### Implementacja klasy BroadcastReceiver

Sama implementacja własnego BroadcastReceivera do prostego rozszerzenia abstrakcyjnej klasy **BroadcastReceiver**.

|  |
| --- |
| 1 public abstract class AbstractBroadcastReceiverTaskObject  2 **extends** BroadcastReceiver **implements** TaskObject **{**  3 protected static Intent responseIntent**;**  4 protected Boolean activated**;**  5 @Override  6 public void onReceive**(**Context context**,** Intent intent**)** **{**  7 Map**<**Boolean**,** Set**<**Intent**>>** activationSet **=**  8 receive**(**context**,** intent**);**  9 **for** **(**Map**.**Entry**<**Boolean**,** Set**<**Intent**>>** activationEntry **:**  10 activationSet**.**entrySet**())** **{**  11 **for** **(**Intent intentFromEntry **:**  12 activationEntry**.**getValue**())** **{**  13 intentFromEntry**.**putExtra**(**  14 ConstanceFieldHolder**.**EXTRA\_TRIGGER\_ACTIVATED**,**  15 activationEntry**.**getKey**());**  16 intentFromEntry**.**putExtra**(**  17 ConstanceFieldHolder**.**EXTRA\_CLASS\_NAAME**,**  18 getClass**().**getName**());**  19 context**.**sendBroadcast**(**intentFromEntry**);**  20 **}**  21 **}**  22 **}**  23 protected abstract Map**<**Boolean**,** Set**<**Intent**>>** receive**(**  24 Context context**,** Intent intent**);** |

Listing Przykładowa implementacja BroadcastReceivera

Jak można zauważyć na powyższym kodze źródłowym podczas rozszerzania klasy **BroadcastReceiver** programista musi napisać własną implementacje metody onReceive(). Funkcja ta przyjmuje dwa paramenty:

* Obiekt klasy **Context** – będący reprezentacją wątku widoku lub serwisu na rzecz, którego działa odbiornik.
* Intencje - zawierają akcje oraz dane dodatkowe, dzięki któremu został uruchomiony.

### Rejestracja odbiornika

Tak jak aktywności oraz serwisy, BroadcastReceivery muszą zostać zarejestrowane przez system. Niemniej jednak w odróżnieniu od dwóch pierwszych komponentów Androida, odbiornik używany przez program nie musi powiadamiać o swoim istnieniu w manifeście aplikacji (*statycznie*), ale także można to zrobić przy pomocy w samym jej kodzie (*dynamicznie*).

|  |
| --- |
| 1 <receiver android:name=  "org.karolgurecki.autotask.service.StartUpBroadcastReceiver">  2 <intent-filter>  3 <action android:name=  "android.intent.action.ACTION\_EXTERNAL\_APPLICATIONS\_AVAILABLE"/>  4 <action android:name=  "android.intent.action.BOOT\_COMPLETED"/>  5 </intent-filter>  6 </receiver> |

Listing Przykład rejestracji BroadcastReceivera z użyciem AdroidManifest.xml

Listing 10 przedstawia fragment AndroidManifest.xml odpowiedzialny   
za rejestracje odbiornika w systemie. Dzięki zdefiniowaniu BroadcastReceivera   
w tym pliku, można powiadomić system o istnieniu tego BroadcastReceivera bez uruchamiania kontekstu aplikacji na rzeczy, której jest tworzony. Przy pomocy tego podejścia są one rejestrowane od razu po pomyślnym uruchomieniu Androida, czyli istnieje możliwość stworzenia odbiornika, który będzie czekał   
na to wydarzenie (tak jak BroadcastReceiver przedstawiony na listingu 10).

Drugim sposobem powiadamiania Androida o istnieniu odbiornika jest użycie do tego kodu aplikacji. Aby go użyć trzeba wykonać następujące kroki:

1. Stworzyć **IntentFilter** zawierającą akcję, które chcemy wygłosić
2. Na koniec metodę registerReceiver() używając obiektu typu **Context**

Powyższe kroki można zaobserwować na listingu 11.

|  |
| --- |
| 1 context**.**registerReceiver**(**  2 taskObject**.**getBroadcastReceiver**(),**  3 taskObject**.**getIntentFilter**());** |

Listing Rejestracja nowego odbiornika w kodze aplikacji

Przy dynamicznej rejestracji odbiornika trzeba pamiętać o jego   
od rejestrowaniu. Jak to nie zostanie wykonane może dojść do wycieku pamięci   
na urządzeniu. Służy do tego metoda unrigisterReceiver() pobierająca referencje do **BroadcastReceivera**, którego chcemy dezaktywować.

### Wysyłanie wiadomości oraz cykl życia odbiornika

Do nadawania wiadomości służy metoda sendBroadcast() z klasy **Context**. Poniższy listing prezentuje budowanie intencji wraz z dodatkowymi danymi oraz rozsyłanie go przy pomocy powyższej funkcji.

|  |
| --- |
| 1 Intent taskObjectIntent **=** TaskObject**.**getIntent**();**  2 taskObjectIntent**.**putExtra**(**  3 ConstanceFieldHolder**.**TASK\_HOLDER\_NAME\_EXTRA**,** name**);**  4 context**.**sendBroadcast**(**taskObjectIntent**);** |

Listing Wysyłanie wiadomości o zdarzeniu

Podczas używania BroadcastReceiverów warto wiedzieć jak one się zachowują po zakończeniu wykonywania się metody onReceiver(). Domyślnie odbiorniki nie mogą wykonywać obliczeń w innym wątku niż ten, z którego  
są uruchamiane, a po ich zakończeniu system może uwolnić zasoby przez nią używane. W Androidzie 3.0 wprowadzono metodę goAsync() pozwalającą  
na kontynuowanie obliczeń asynchronicznie. Dzięki tej funkcji system będzie mógł usunąć danego BroadcastReceivera z pamięci, kiedy zostanie wywołana metoda finish() z obiekty klasy **PendingResult** otrzymanego po wywołaniu funkcji goAsync().

## Lokalne BroadcastReceivery

Podczas tworzenia aplikacji zdarza się czasami, że programista nie chce lub nie powinien rozgłaszać albo odbierać pewnym informacji z całego urządzenia. Najczęściej jest to spowodowane względami bezpieczeństwa. Wysyłanie wrażliwych informacji w kierunku wszystkich komponentów w systemie może doprowadzić do ich wycieku, a odbieranie wszystkich transmisji danego wydarzenia do nieautoryzowanego sterowania określonego komponentu przez inny zewnętrzny.

Z powodu przedstawionego powyżej firma Google postanowiła w **Android Support Library** dodać klasę **LocalBroadcastManager**. Jej zadaniem jest zarządzanie nadawaniem oraz odbieraniem informacji przekazywanych wewnątrz aplikacji.

Sam proces rejestrowania oraz wysyłania informacji odbywa się bardzo podobnie do standardowego korzystania z tego mechanizmu. Trzeba pobrać nową instancje **LocalBroadcastManager** poprzez getInstance(), następnie używając otrzymanego obiektu można tak jak w przypadku globalnych zarządzać odbiornikami oraz wysyłać wiadomości. Nazwy metod oraz ich paramenty  
są identyczne do dostępnych w klasie **Context**. Poniższy kod źródłowy przedstawia otrzymanie instancji klasy **LocalBroadcastManager** oraz zarejestrowanie, natomiast listing 14 ukazuje wysyłanie wiadomości przy użyciu tego samego obiektu.

|  |
| --- |
|  |

Listing Otrzymanie nowej instancji LocalBroadcastManager oraz rejestracja lokalnego odbiornika

|  |
| --- |
|  |

Listing Nadanie lokalnej wiadomości

## BroadcastReceivery udostępniane przez Androida

### Wydarzenia dostępne w klasie Intent do powszechnego użytkowania

Poniższa tabela przedstawia wybrane akcje z klasy Intent. Wszystkie nazwy akcji reprezentant nazwy zmiennych typu **String**, którą trzeba podać przy stworzeniu obiektu **IntentFilter** potrzebnego do zarejestrowania odbiornika.

Tabela Wydarzenia z klasy Intent dostępne do korzystania w aplikacjach innych niż systemowe

|  |  |
| --- | --- |
| **Nazwa akcji** | **Opis** |
| ACTION\_AIRPLANE\_MODE\_CHANGED | Wysyłane, kiedy użytkownik włączył lub wyłączył tryb samolotowy. Posiada informacje o stanie, w którym znajduje się ten tryb. |
| ACTION\_BATTERY\_  CHANGED | Zawiera informacje o stanie baterii. Ten Intent nie może być użyty w AndroidManifest.xml |
| ACTION\_BATTERY\_LOW | Wysyłany, kiedy baterie w urządzeniu osiągnęła niski poziom naładowania. |
| ACTION\_BOOT\_  COMPLETED | Aplikacja otrzymuje tą transmisje, kiedy system zakończył uruchamianie się. Musi posiadać uprawnienie **RECEIVE\_BOOT\_COMPLETED**. |
| ACTION\_CONFIGURATION\_CHANGED | Wysyłany, kiedy na urządzeniu zmieniła się konfiguracja jak na przykład zmiana jeżyka czy orientacji. Większość programów nie musi go implementować, ponieważ system sam zastosuje zmiany poprzez restart aplikacji. |
| ACTION\_HEADSET\_PLUG | Informuje, że słuchawki przewodowe zostały podpięte lub odłączone od urządzenia. Zawiera dane o ich stanie, nazwie oraz czy posiadają mikrofon. |
| ACTION\_NEW\_OUTGOING\_CALL | Wysyłany, kiedy z urządzenia wychodzi połączenie głosowe. Zawiera informacje  o numerze, który został wybrany |
| ACTION\_POWER\_  CONNECTED | Informuje, że ładowarka została podłączona  do urządzenia. |
| ACTION\_POWER\_DISCONNECTED | Wysyłany, kiedy ładowarka została odłączona od urządzenia. |
| ACTION\_REBOOT | Oznacza, że urządzenie zostanie zrestartowany. Jest używany tylko przez komponenty systemowe. |
| ACTION\_SCREEN\_OFF | Oznacza, że urządzenie jest wstanie uśpienia. Warto zaznaczyć, iż nie koniecznie ekran urządzenia jest wyłączony. Aby sprawdzić stan wyświetlacza trzeba użyć metody getState() z klasy **Display**. |
| ACTION\_SHUTDOWN | Wysyłany, kiedy urządzenie się wyłącza. |
| ACTION\_TIME\_CHANGED | Informuje, że czas został zmieniany  w ustawieniach urządzenia. |
| ACTION\_TIME\_TICK | Wysyłany, co minutę. Informuje o zmianie czasu. |

### Przykłady wydarzenie z innych klas dostępnych w API Androida

Poniższa tabela przedstawia wybrane akcje z innych klas API Androida.   
Tak jak w poprzednim podrozdziale wszystkie nazwy akcji reprezentant nazwy zmiennych typu **String** (wraz z nazwą klasy, z której pochodzi), którą trzeba podać przy stworzeniu obiektu **IntentFilter** potrzebnego do zarejestrowania odbiornika.

Tabela Wybrane wydarzenia z innych klasy Android API

|  |  |
| --- | --- |
| **Nazwa akcji** | **Opis** |
| BluetoothDevice.ACTION\_BOND\_STAGE\_CHANGED | Informuje, że stan połączenia  z urządzeniem zewnętrznym przez interfejs Bluetooth się zmienił. |
| BluetoothDevice.ACTION\_FOUND | Wysyłany, kiedy nowe urządzenia zostały wykryte. |
| BluetoothDevice. ACTION\_PAIRING\_REQUEST | Używany do wysłania prośby o połączenie się z innym urządzeniem przez interfejs Bluetooth. Wymaga, aby aplikacja posiadała uprawnienie **BLUETOOTH\_ADMIN**. |
| BluetoothAdapter. ACTION\_LOCAL\_NAME\_  CHANGED | Informuje, że nazwa urządzenia widziana przez inne urządzenia Bluetooth została zmieniona. |
| BluetoothAdapter. ACTION\_SCAN\_MODE\_  CHANGED | Wysyłany, kiedy tryb wyszukiwania nowych urządzeń Bluetooth się zmienił. Zawiera dane o aktualnym stanie oraz poprzednim. |
| BluetoothAdapter. ACTION\_STATE\_CHANGED | Informuje, że moduł Bluetooth  na urządzeniu został włączony lub wyłączony. Zawiera dane o aktualnym stanie oraz poprzednim. |
| NfcAdapter. ACTION\_ADAPTER\_STATE\_  CHANGED | Informuje, że moduł NFC na urządzeniu został włączony lub wyłączony. |
| Telephony.Sms.Intents. SIM\_FULL\_ACTION | Informuje, że pamięć SIM dla krótkich wiadomości jest pełna. |
| Telephony.Sms.Intents. SMS\_DELIVER\_ACTION | Wysyłany, kiedy wiadomość SMS została dostarczona do nadawcy. |
| Telephony.Sms.Intents. WAP\_PUSH\_RECEIVED\_  ACTION | Oznacza, że urządzenie otrzymało wiadomość typu WAP PUSH. |
| WifiManager. NETWORK\_STATE\_CHANGED\_ACTION | Informuje o zmianie stanu połączenia  z punktem dostępowym Wi-Fi. |
| WifiManager. WIFI\_STATE\_CHANGED\_  CTION | Informuje, że moduł Wi-Fi na urządzeniu został włączony lub wyłączony. Zawiera dane o aktualnym stanie oraz poprzednim. |
| UsbManager. ACTION\_USB\_ACCESSORY\_  ATTACHED | Wysyłany, kiedy urządzanie zostało podłączone z innym przez interfejs USB w trybie *accessory*. |
| UsbManager. ACTION\_USB\_DEVICE\_  ATTACHED | Wysyłany, kiedy urządzanie zostało podłączone z innym przez interfejs USB w trybie *host*. |

## Użycie BroadcastReceivera oraz intencji na przykładzie zmian stanu modułu Wi-Fi

# Bibliografia

1. *Worldwide Smartphone Market Grows 28.6% Year Over Year in the First Quarter of 2014, According to IDC* [online]. [Framingham]: IDC Corporate USA, 2014 [dostęp na dzień 25.06.2014], dostępny w internecine: <http://www.idc.com/getdoc.jsp?containerId=prUS24823414>
2. *Api Guides* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 25.06.2014], dostępny w Internecie:

<http://developer.android.com/guide/index.html>

1. *Training* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 25.06.2014], dostępny w Internecie:

<http://developer.android.com/training/index.html>

1. *Android BroadcastReceiver - Tutorial* [online]. [Hamburg]: Vogella, 2013 [dostęp na dzień 25.06.2014], dostępny w Internecie:

<http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html#resources_general>

1. Meier, R. *Professional Android 4 Application Development*, Wyd. 3, Indianapolis,2013, ISBN 978-1118102275
2. Conder Shane, Darcey Lauren. *Android. Programowanie aplikacji   
   na urządzenia przenośne*, Wyd. 2, Helion, Gliwice, 2012, ISBN 978-83-246-3349-4
3. *Google Android prototypes debut at MWC* [online]. [New York]: CBS Interactive Inc., 2008 [dostęp na dzień 01.07.2014], dostępny w Internecie: <http://www.cnet.com/news/google-android-prototypes-debut-at-mwc/>
4. *A brief history of Android phones* [online]. [San Francisco]: CBS Interactive Inc., 2011 [dostęp na dzień 01.07.2014], dostępny w Internecie: <http://www.cnet.com/news/a-brief-history-of-android-phones/>
5. *Google I/O: Android stands at one billion active users and counting* [online]. [San Francisco]: CBS Interactive Inc., 2014 [dostęp na dzień 01.07.2014], dostępny w Internecie:

<http://www.zdnet.com/google-io-android-stands-at-one-billion-active-users-and-counting-7000030881/#ftag=RSSf468ffe?ref=synergymx>

1. *Q1 2014 Smartphone OS Results: Android Dominates High Growth Developing Markets* [online]. [Londyn]: Allied Business Intelligence, Inc. 2014 [dostęp na dzień 01.07.2014], dostępny w Internecie:

<https://www.abiresearch.com/press/q1-2014-smartphone-os-results-android-dominates-hi>

1. *Activieties* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 2.07.2014], dostępny w Internecie:

<http://developer.android.com/guide/components/activities.html>

1. *Android Intents - Tutorial* [online]. [Hamburg]: Vogella, 2014 [dostęp na dzień 16.07.2014], dostępny w Internecie:

<http://www.vogella.com/tutorials/AndroidIntent/article.html>

1. *Intent* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://developer.android.com/reference/android/content/Intent.html>

1. *Uniform Resource Identifiers (URI): Generic Syntax* [online]. [Wiehle Avenue]: Internet Society, 1998 [dostęp na dzień 29.07.2014], dostępny w Internecie:

<http://tools.ietf.org/pdf/rfc2396.pdf>

1. *Android BroadcastReceiver - Tutorial* [online]. [Hamburg]: Vogella, 2013 [dostęp na dzień 30.07.2014], dostępny w Internecie:

<http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html>

1. *BluetoothAdapter* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

[http://developer.android.com/reference/android/bluetooth/BluetoothAdapter](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html)

1. *BluetoothManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

[http://developer.android.com/reference/android/content/BluetoothManager](http://developer.android.com/reference/android/content/BluetoothManager.html)

1. *Telephony.Sms.Intents* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<https://developer.android.com/reference/android/provider/Telephony.Sms.Intents.html>

1. *WifiManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://developer.android.com/reference/android/net/wifi/WifiManager.html>

1. *UsbManager* [online]. [Mountain View]: Google Inc., 2014 [dostęp na dzień 20.07.2014], dostępny w Internecie:

<http://developer.android.com/reference/android/hardware/usb/UsbManager.html>

nn

todo popraw spisy!!!!!!!!!!!!!!!

# Spis rysunków

1. [Przykładowy zrzut ekranu z Intelij Android Studio 9](#_Toc394492378)
2. [Cykl życia aktywności. Źródło: [11] 12](#_Toc394492379)
3. [Okno wyboru domyślnej przeglądarki w systemie 22](#_Toc394492380)

# Spis tabel

1. [Przykładowe znaczniki, które są dostępne w Android Manifest 9](#_Toc394685867)
2. [Przykładowe uprawnienie, które udostępnia Android 13](#_Toc394685868)
3. [Wydarzenia z klasy Intent dostępne do korzystania w aplikacjach innych niż systemowe 30](#_Toc394685869)
4. [Wybrane wydarzenia z innych klasy Android API 32](#_Toc394685870)

# Spis kodów źródłowych

1. [Przykładowy plik AndrodManifest.xml 11](#_Toc394685650)
2. [Tworzenie Activity przy pomocy kodu 12](#_Toc394685651)
3. [Przykładowy plik XML zawierający informacje na temat układu widoku 16](#_Toc394685652)
4. [Przykładowy plik strings.xml 17](#_Toc394685653)
5. [Fragment StartUpService.java 18](#_Toc394685654)
6. [Definicja StartUpService w AndroidManifest.xml 19](#_Toc394685655)
7. [Przykładowe jednoznaczne uruchomienie aktywności 21](#_Toc394685656)
8. [Wyświetlenie strony WWW w domyślnej przeglądarce 21](#_Toc394685657)
9. [Przykładowa implementacja BroadcastReceivera 27](#_Toc394685658)
10. [Przykład rejestracji BroadcastReceivera z użyciem AdroidManifest.xml 28](#_Toc394685659)
11. [Rejestracja nowego odbiornika w kodze aplikacji 28](#_Toc394685660)
12. [Wysyłanie wiadomości o zdarzeniu 29](#_Toc394685661)
13. [Otrzymanie nowej instancji LocalBroadcastManager oraz rejestracja lokalnego odbiornika 30](#_Toc394685662)
14. [Nadanie lokalnej wiadomości 30](#_Toc394685663)

1. **Smartphone** (z ang. *Inteligentny telefon*) – urządzenie elektroniczne, które można zmieścić   
   w kieszenie posiadające możliwość wykonywania połączeń głosowych oraz pozwalające   
   na instalowanie i uruchamianie oprogramowania firm trzecich. [↑](#footnote-ref-1)
2. **API** (skrót od *Application Programming Interface* – z ang. *interfejs programowania aplikacji*) – podprogramy, struktury danych, klasy obiektów, które udostępnia dany program, biblioteka, system operacyjny w celu komunikacji w innym oprogramowaniem. [↑](#footnote-ref-2)
3. Tutorial (z ang. *Samouczek*) – zestaw instrukcji pozwalający na łatwe nauczenie się określonego zagadnienia. [↑](#footnote-ref-3)
4. **IDE** (skrót od *integrated development environment*, z ang. *zintegrowane środowisko programistyczne*) – jedna lub wiele aplikacji pozwalających na edycje, budowanie oraz testowanie kodu programu przez programistę [↑](#footnote-ref-4)
5. **SDK** (skrót od *Software Development Kit*) – zastaw narzędzi ułatwiający tworzenie programów na dane środowisko [↑](#footnote-ref-5)
6. **LIFO** (skrót od *last in, firsrt out*) – algorytm kolejkowania elementów na liście jedno wymiarowej. Oznacza, że element, który jako ostatni został dodany do tej kolekcji, będzie zdjęty jako pierwszy przy próbie pobrania z niej danych. [↑](#footnote-ref-6)
7. **Best practice** (z ang. *Najlepszy sposób*) – ten wzór w programowaniu oznacza, zalecany sposób wykonania często spotykanych problemów [↑](#footnote-ref-7)
8. String (z ang. *łańcuch znaków*) – typ zmiennej w języku Java, który może przechowywać całe ciągi znaków. [↑](#footnote-ref-8)
9. **URI** (skrót od ***Uniform Resource Identifier***) – standard zapisu odnośników, w postaci ciągu znaków, do zasobów, w celu ich łatwej identyfikacji. Zostaw stworzony przez Internet Society [14]. [↑](#footnote-ref-9)
10. Setter – zwyczajowo przyjęty typ metody w języku Java pozwalający na ustawianie wartości zmiennych w obiekcie. [↑](#footnote-ref-10)