Практическая работа по NFC

Методические материалы

Отправка файла на другое устройство

В этом материале мы рассмотрим, как передавать большие файлы на другое устройство, используя Android Beam. Android Beam — это технология Near Field Communication (NFC), представленная в Android 4.0, которая позволяет приложениям совместно использовать информацию через NFC, если они находятся поблизости друг от друга.

Android Beam поддерживает передачу сообщений через NFC, если два устройства находятся в диапазоне действия. Устройства, которые находятся друг от друга на расстоянии 4 см, могут обмениваться данными с помощью Android Beam.

Для отправки файлов нужно получить разрешение на использование NFC и внешнего хранилища, убедиться, что устройство поддерживает NFC и предоставить URI файлов.

Механизм передачи файлов Android Beam имеет следующие требования:

1. Android Beam для передачи больших файлов доступен только начиная с Android 4.1 (API 16).
2. Файлы для передачи должны находиться на внешнем хранилище. Больше об этом читайте в разделе Внешнее хранилище
3. Каждый файл, который вы хотите передать, должен быть доступен для чтения всем. Установить такие права на чтение можно с помощью метода File.setReadable(true, false).
4. Необходимо предоставить URI файла со схемой file, который вы хотите передать. Android Beam не может работать с URI типа content, сгенерированного с помощью метода FileProvider.getUriForFile.

Добавляем функцию в манифест

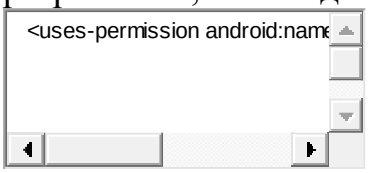
Сначала добавьте в файл манифеста необходимые разрешения и функции, которые потребуются приложению.

Требуемые разрешения

Чтобы разрешить приложению передавать файлы с внешнего хранилища через NFC, необходимо установить следующие разрешения в манифесте:

NFC

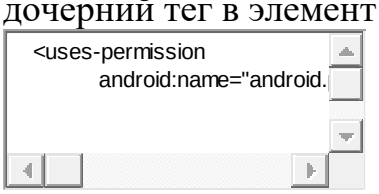
Разрешает приложению отправлять данные через NFC. Чтобы добавить разрешение, необходимо добавить в элемент <manifest> дочерний:




```
1 <uses-permission android:name="android.permission.NFC" />
```

READ_EXTERNAL_STORAGE

Разрешает приложению читать из внешнего хранилища. Добавьте дочерний тег в элемент <manifest>:



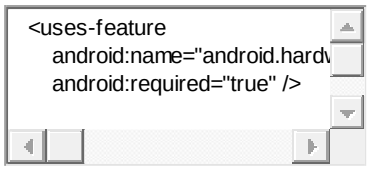

```
1 <uses-permission
2         android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Примечание: на момент Android 4.2.2(API 17), это разрешение не применяется. В будущих версиях это разрешение может понадобиться, чтобы читать из внешнего хранилища. Чтобы не беспокоиться в будущем, просто добавьте это разрешение уже сейчас.

Добавляем функции NFC

Добавьте дочерний элемент <uses-feature> в <manifest>, чтобы включить использование NFC приложением. Установите атрибут android:required равным true, чтобы указать, что приложение не будет работать, если NFC отсутствует.

Элемент <uses-feature> может выглядеть так:




```
1 <uses-feature
2     android:name="android.hardware.nfc"
3     android:required="true" />
```

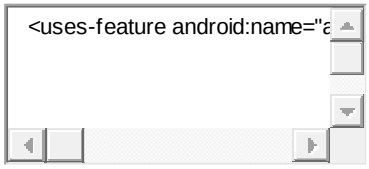
Если ваше приложение использует NFC в качестве дополнительной опции и прекрасно работает без него, установите атрибут android:required равным false и проверяйте наличие NFC в коде.

Добавляем функцию передачи файлов Android Beam

Android Beam доступен с версии Android 4.1 (API 16), поэтому если использование Android Beam для передачи файлов является ключевой функцией вашего приложения, установите атрибут android:minSdkVersion="16".

Проверяем поддержку передачи файлов Android Beam

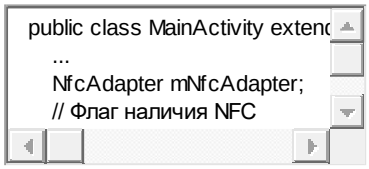
Чтобы указать, что использование NFC является опциональным, используйте следующий код:




```
1 <uses-feature android:name="android.hardware.nfc" android:required="false" />
```

Если вы установили атрибут android:required="false", проверяйте поддержку NFC программно.

Чтобы программно проверить поддержку Android Beam для передачи файлов, вызовите метод PackageManager.hasSystemFeature() с аргументом FEATURE_NFC. Затем проверьте значение SDK_INT. Если передача файлов Android Beam поддерживается, получите экземпляр NFC контроллера. Например так:



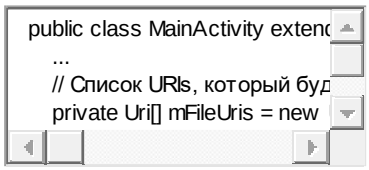

```
1  public class MainActivity extends Activity {
2      ...
3      NfcAdapter mNfcAdapter;
4      // Флаг наличия NFC
5      boolean mAndroidBeamAvailable  = false;
6      ...
7      @Override
8      protected void onCreate(Bundle savedInstanceState) {
9          ...
10         // Если NFC не поддерживается устройством
11         if
12 (!PackageManager.hasSystemFeature(PackageManager.FEATURE_NFC)) {
13             /*
14              * Запрещаем все функции, которые используются NFC
```

```
15            * Например, пункты меню, кнопки, и.т.п
16            */
17           ...
18       // Если не поддерживается передача файлов Android Beam
19       } else if (Build.VERSION.SDK_INT <
20               Build.VERSION_CODES.JELLY_BEAN_MR1) {
21           // Не продолжаем, если Android Beam недоступно
22           mAndroidBeamAvailable = false;
23           /*
24            * Запрещаем функции, которым нужен Android Beam
25            */
26           ...
27       // Если Android beam доступен, продолжаем
28       } else {
29       mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
30       ...
31       }
32   }
33   ...
}
```

Создание метода обратного вызова для передачи файлов

После проверки поддержки Android Beam, создайте метод обратного вызова, который система будет вызывать при попытке передать файл через NFC. Этот метод будет возвращать массив Uri объектов. Android Beam передаст файлы по указанным Uri на другое устройство.

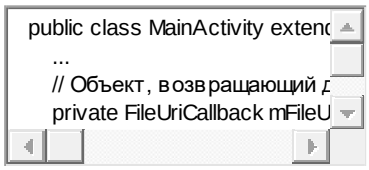
Чтобы добавить метод обратного вызова, реализуйте интерфейс NfcAdapter.CreateBeamUrisCallback и метод createBeamUris(). Пример, как это сделать:




```
1  public class MainActivity extends Activity {
2      ...
3      // Список URIs, который будет передан Android Beam
4      private Uri[] mFileUris = new Uri[10];
5      ...
6      /**
7       * Метод, который Android Beam вызывает для получения
8       * файлов
9       */
```

```
10    private class FileUriCallback implements
11            NfcAdapter.CreateBeamUrisCallback {
12        public FileUriCallback() {
13        }
14        /**
15         * Создаем URIs, чтобы передать на другие устройства
16         */
17        @Override
18        public Uri[] createBeamUris(NfcEvent event) {
19            return mFileUris;
20        }
21    }
22    ...
23 }
```

После реализации интерфейса, передайте экземпляр созданного класса в метод setBeamPushUrisCallback(). Например:

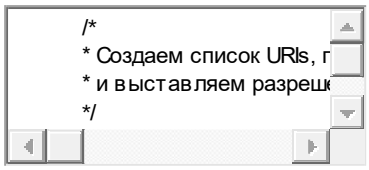



```
1   public class MainActivity extends Activity {
2       ...
3       // Объект, возвращающий доступные файлы
4       private FileUriCallback mFileUriCallback;
5       ...
6       @Override
7       protected void onCreate(Bundle savedInstanceState) {
8           ...
9           // Android Beam доступен
10          ...
11          mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
12          /*
13           * Создаем объект FileUriCallback
14           */
15          mFileUriCallback = new FileUriCallback();
16          // Устанавливаем метод обратного вызова для запроса URI.
17          mNfcAdapter.setBeamPushUrisCallback(mFileUriCallback,this);
18          ...
19      }
20      ...
21  }
```

Примечание: можно также напрямую передавать URI в NFC фреймворк из вашего приложения с помощью экземпляра класса NfcAdapter. Этот способ подходит для случаев, когда вы хотите создать URI до того, как произойдет касание NFC устройств. Подробнее об этом в разделе NfcAdapter.setBeamPushUris().

Выбираем файлы для отправки

Для передачи одного или нескольких файлов на другое устройство с NFC, получите файловые URI (со схемой file) для каждого файла и добавьте их в массив Uri объектов. Также для передачи нужны постоянные права на чтение файла. Например, получим URI из имени файла и добавим его в массив:




```
1        /*
2         * Создаем список URIs, получаем объект File
3         * и выставляем разрешения
4         */
5        private Uri[] mFileUris = new Uri[10];
6        String transferFile = "transferimage.jpg";
7        File extDir = getExternalFilesDir(null);
8        File requestFile = new File(extDir, transferFile);
9        requestFile.setReadable(true, false);
10       // Получаем URI для файла и добавляем его в массив
11       fileUri = Uri.fromFile(requestFile);
12       if (fileUri != null) {
13           mFileUris[0] = fileUri;
14       } else {
15           Log.e("My Activity", "No File URI available for file.");
16       }
```

Получение файла с другого устройства

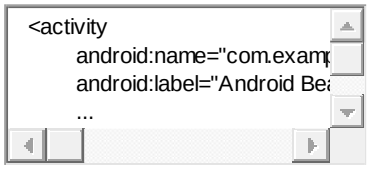
Android Beam помещает полученные файлы в специальную директорию. Он также сканирует файлы медиа-сканером и добавляет найденные медиа-файлы в MediaStore провайдер. В уроке рассказывается, что делать после завершения копирования файлов и как найти полученные файлы на устройстве.

Отображение данных

После завершения передачи, Android Beam передает уведомление, включающее намерение с действием ACTION_VIEW, MIME тип и URI первого файла. При клике на уведомление, намерение отправляется в систему. Чтобы ваше приложение могло обработать это намерение, добавьте элемент <intent-filter> в элемент <activity> явления, которое должно отвечать на запрос. В элемент <intent-filter> добавьте следующие дочерние элементы:

<action android:name=”android.intent.action.VIEW” />
Указывает на действие ACTION_VIEW намерения.
<category android:name=”android.intent.category.CATEGORY_DEFAULT” />
Указывает, что намерение не относится к категории явных
<data android:mimeType=”mime-type” />
Указывает MIME тип. Указывайте только те типы, которые может обработать ваше приложение.

Пример фильтра намерений для явления com.example.android.nfctransfer.ViewActivity:

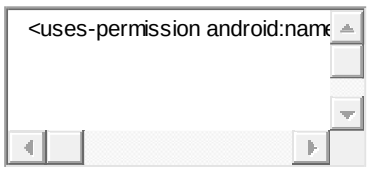



```
1  <activity
2          android:name="com.example.android.nfctransfer.ViewActivity"
3          android:label="Android Beam Viewer" >
4          ...
5          <intent-filter>
6              <action android:name="android.intent.action.VIEW"/>
7              <category android:name="android.intent.category.DEFAULT"/>
8              ...
9          </intent-filter>
10     </activity>
```

Примечание: передача файлов Android Beam это не единственный источник намерений с действием ACTION_VIEW. Другие приложения могут также передавать намерение с таким действием. Обработка подобных ситуаций обсуждается в разделе: Получение директории из URI данных

Запрос разрешения на файл

Чтобы прочитать принятый файл, необходимо запросить права на чтение READ_EXTERNAL_STORAGE, например:




```
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Если вы хотите скопировать принятые файлы в собственное внешнее хранилище приложения, запросите права WRITE_EXTERNAL_STORAGE. Права на запись включают в себя также права и на чтение.

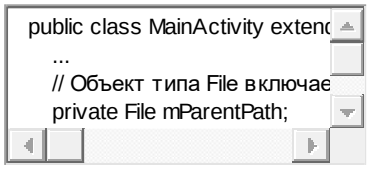
Примечание: на момент Android 4.2.2 (API 17) права на чтение внешнего хранилища запрашиваются по желанию пользователя. В будущих версиях платформы, права могут потребоваться во всех случаях. Поэтому просто добавьте запрос прав уже сейчас, чтобы не беспокоиться об этом в будущем.

Так как приложение имеет полный контроль над внутренним хранилищем, вам не надо запрашивать права, чтобы записать в него принятый файл.

Получаем директорию принятых файлов

Android Beam размещает все переданные за раз файлы в одну директорию. Намерение, которое передает Android Beam в уведомлении, содержит URI первого переданного файла. Однако не забывайте, что приложение может получить намерение с действием ACTION_VIEW не только от Android Beam, но и от других приложений. Чтобы решить как обрабатывать входящее намерение, необходимо проверить схему и authority.

Чтобы получить схему для URI, вызовите метод Uri.getScheme(). Следующий пример показывает как определить схему и соответствующий обработчик для URI:

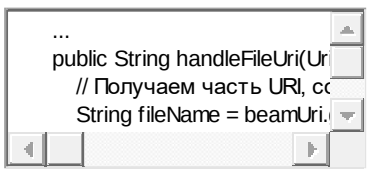



```
public class MainActivity extends Activity {
    ...
    // Объект типа File включает также путь до принятых файлов
    private File mParentPath;
    // Входящее намерение
    private Intent mIntent;
    ...
    /*
     * Вызывается из метода onNewIntent() для явления SINGLE_TOP
```

```
10     * или из метода onCreate() для нового явления. В случае onNewIntent(),
11     * не забудьте вызвать setIntent()
12     *
13     */
14    private void handleViewIntent() {
15        ...
16        // Получаем действие намерения
17        mIntent = getIntent();
18        String action = mIntent.getAction();
19        /*
20         * Для действия ACTION_VIEW, показываем данные.
21         * Получаем URI.
22         */
23        if (TextUtils.equals(action, Intent.ACTION_VIEW)) {
24            // Получаем URI из намерения
25            Uri beamUri = mIntent.getData();
26            /*
27             * Проверяем тип URI, получив его схему
28             */
29            if (TextUtils.equals(beamUri.getScheme(), "file")) {
30                mParentPath = handleFileUri(beamUri);
31            } else if (TextUtils.equals(
32                    beamUri.getScheme(), "content")) {
33                mParentPath = handleContentUri(beamUri);
34            }
35        }
36        ...
37    }
38    ...
39 }
```

Получение директории из URI типа file

Если входящее намерение включает URI со схемой file, этот URI включает в себя абсолютный путь до файла и его имя. Как было сказано ранее, в Android Beam все файлы текущий передачи находятся в одном и том же каталоге. Чтобы получить путь до каталога, необходимо взять полный путь из URI (URI это вся строка, кроме префикса file:). После этого используйте объект типа File, чтобы получит адрес каталога:




```
1     ...
```

```
2     public String handleFileUri(Uri beamUri) {
3         // Получаем часть URI, содержащую путь до файла
4         String fileName = beamUri.getPath();
5         // Создаем объект File для данного файла
6         File copiedFile = new File(fileName);
7         // Получаем строку с директорий, в которой находится файл
8         return copiedFile.getParent();
9     }
10    ...
```

Получение директории из URI типа content

Если входящее намерение содержит URI типа content, директория и имя файла хранятся в поставщике содержимого MediaStore. Сделать вывод, что перед нами URI типа content можно проверив значение authority. URI типа content для провадера MediaStore может быть получено из Android Beam или любого другого приложения, но в обоих случаях вы можете получить директорию и имя файла для этого URI.

Вы можете также получить входящее намерение с действием ACTION_VIEW, включающее URI типа content для другого поставщика содержимого, помимо MediaStore. В данном случае URI не содержит значение authority для MediaStore и такой URI обычно не ссылается на директорию.

Примечание: в Android Beam вы получите URI типа content в том случае, если первый принятый файл имеет MIME тип “audio/*”, “image/*” или “video/*”, то есть является медиа-файлом. Android Beam индексирует все такие файлы с помощью медиа-сканера. Медиа-сканер записывает результаты индексации в поставщик данных MediaStore, а затем передает URI первого файла обратно в Android Beam. Это будет URI, который вы получили в уведомлении. Получить каталог первого файла можно из MediaStore, используя URI со схемой content.

Определение поставщика данных

Чтобы получить каталог из URI типа content, определите тип поставщика содержимого, связанного с URI, с помощью метода Uri.getAuthority(). Метод возвращает значение authority, которое означает следующее:

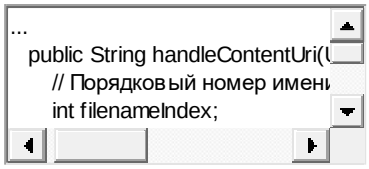
MediaStore.AUTHORITY
: URI файла, который отслеживается MediaStore. Получите полный путь до файла из MediaStore, а директорию вычислите из это пути.

Любое другое значение authority

URI любого другого поставщика содержимого. Нельзя получить директорию по URI.

Чтобы получить директорию при поставщике содержимого MediaStore, выполните запрос, в котором укажите столбец MediaColumns.DATA, а в качестве аргумента передайте URI. Возвращенный курсор (Cursor) включает полный путь и имя файла для заданного URI. По этому пути расположены также остальные файлы, принятые вместе с текущим.

В следующем примере показано как проверить authority у URI и получить путь и имя файла:




```
1  ...
2      public String handleContentUri(Uri beamUri) {
3          // Порядковый номер имени файла в курсоре
4          int filenameIndex;
5          // Объект File для имени файла
6          File copiedFile;
7          // Файл из MediaStore
8          String fileName;
9          // Проверяем authority у URI
10         if (!TextUtils.equals(beamUri.getAuthority(),
11 MediaStore.AUTHORITY)) {
12             /*
13              * Какие-то действия для других поставщиков
14              */
15         // Иначе действия для MediaStore
16         } else {
17             // Получаем столбец с именем файла
18             String[] projection = { MediaStore.MediaColumns.DATA };
19             Cursor pathCursor =
20                     getContentResolver().query(beamUri, projection,
21                     null, null, null);
22             // Проверяем, что курсор существует
23             if (pathCursor != null &&
24                     pathCursor.moveToFirst()) {
25                 // Получаем порядковый номер столбца
26                 filenameIndex = pathCursor.getColumnIndex(
27                         MediaStore.MediaColumns.DATA);
28                 // Получаем полный путь до файла
29                 fileName = pathCursor.getString(filenameIndex);
```

```
30              // Создаем файловый объект по его имени
31              copiedFile = new File(fileName);
32              // Возвращаем путь до каталога
33              return new File(copiedFile.getParent());
34            } else {
35              //  Запрос не сработал, возвращаем null
36              return null;
37            }
38        }
39    }
      ...
```

Подробная информация о получении данных из поставщиков содержимого находится в разделе Получение данных от поставщиков

Задание

Реализовать приложение обмена большими файлами между мобильными устройствами, применив технологию NFC