Common Intents

An intent allows you to start an activity in another app by describing a simple action you'd like to perform (such as "view a map" or "take a picture") in an `Intent` (https://developer.android.com/reference/android/content/Intent.html) object. This type of intent is called an *implicit* intent because it does not specify the app component to start, but instead specifies an *action* and provides some *data* with which to perform the action.

When you call `startActivity()` (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) or `startActivityForResult()` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) and pass it an implicit intent, the system resolves the intent (https://developer.android.com/guide/components/intents-filters.html#Resolution) to an app that can handle the intent and starts its corresponding `Activity` (https://developer.android.com/reference/android/app/Activity.html). If there's more than one app that can handle the intent, the system presents the user with a dialog to pick which app to use.

This page describes several implicit intents that you can use to perform common actions, organized by the type of app that handles the intent. Each section also shows how you can create an intent filter (https://developer.android.com/guide/components/intents-filters.html#Receiving) to advertise your app's ability to perform the same action.

Caution: If there are no apps on the device that can receive the implicit intent, your app will crash when it calls `startActivity()` (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)). To first verify that an app exists to receive the intent, call `resolveActivity()` (https://developer.android.com/reference/android/content/Intent.html#resolveActivity(android.content.pm.PackageManager)) on your `Intent` (https://developer.android.com/reference/android/content/Intent.html) object. If the result is non-null, there is at least one app that can handle the intent and it's safe to call `startActivity()` (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)). If the result is null, you should not use the intent and, if possible, you should disable the feature that invokes the intent.

If you're not familiar with how to create intents or intent filters, you should first read Intents and Intent Filters (https://developer.android.com/guide/components/intents-filters.html).

To learn how to fire the intents listed on this page from your development host, see Verify Intents with the Android Debug Bridge (#AdbIntents).

Google Voice Actions

Google Voice Actions (https://developers.google.com/voice-actions/) fires some of the intents listed on this page in response to voice commands. For more information, see Intents fired by Google Voice Actions (https://developers.google.com/voice-actions/system/#system_actions_reference).

Alarm Clock

Create an alarm

To create a new alarm, use the `ACTION_SET_ALARM` (https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_ALARM) action and specify alarm details such as the time and message using extras defined below.

(https://developers.google.cc
actions/system/#system_act

Google Voice Actions

- *"set an alarm for 7 am"*

Note: Only the hour, minutes, and message extras are available in Android 2.3 (API level 9) and lower. The other extras were added in later versions of the platform.

Action

`ACTION_SET_ALARM` (https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_ALARM)

Data URI

None

MIME Type

None

Extras

`EXTRA_HOUR`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_HOUR)

The hour for the alarm.

`EXTRA_MINUTES`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_MINUTES)

The minutes for the alarm.

`EXTRA_MESSAGE`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_MESSAGE)

A custom message to identify the alarm.

`EXTRA_DAYS`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_DAYS)

An `ArrayList` (https://developer.android.com/reference/java/util/ArrayList.html) including each week day on which this alarm should be repeated. Each day must be declared with an integer from the `Calendar` (https://developer.android.com/reference/java/util/Calendar.html) class such as `MONDAY` (https://developer.android.com/reference/java/util/Calendar.html#MONDAY).

For a one-time alarm, do not specify this extra.

`EXTRA_RINGTONE`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_RINGTONE)

A `content:` URI specifying a ringtone to use with the alarm, or `VALUE_RINGTONE_SILENT` (https://developer.android.com/reference/android/provider/AlarmClock.html#VALUE_RINGTONE_SILENT) for no ringtone.

To use the default ringtone, do not specify this extra.

`EXTRA_VIBRATE`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_VIBRATE)

A boolean specifying whether to vibrate for this alarm.

`EXTRA_SKIP_UI`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_SKIP_UI)

A boolean specifying whether the responding app should skip its UI when setting the alarm. If true, the app should bypass any confirmation UI and simply set the specified alarm.

Example intent:

```
public void createAlarm(String message, int hour, int minutes) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)
            .putExtra(AlarmClock.EXTRA_MESSAGE, message)
            .putExtra(AlarmClock.EXTRA_HOUR, hour)
            .putExtra(AlarmClock.EXTRA_MINUTES, minutes);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Note:

In order to invoke the `ACTION_SET_ALARM` (https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_ALARM) intent, your app must have the `SET_ALARM` (https://developer.android.com/reference/android/Manifest.permission.html#SET_ALARM) permission:

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SET_ALARM" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Create a timer

To create a countdown timer, use the `ACTION_SET_TIMER`

Google Voice Actions

(https://developers.google.c
actions/system/#system_act

- *"set timer for 5 minutes"*

(https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_TIMER) action and specify timer details such as the duration using extras defined below.

Note: This intent was added in Android 4.4 (API level 19).

Action

`ACTION_SET_TIMER`
(https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_TIMER)

Data URI

None

MIME Type

None

Extras

`EXTRA_LENGTH`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_LENGTH)

The length of the timer in seconds.

`EXTRA_MESSAGE`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_MESSAGE)

A custom message to identify the timer.

`EXTRA_SKIP_UI`
(https://developer.android.com/reference/android/provider/AlarmClock.html#EXTRA_SKIP_UI)

A boolean specifying whether the responding app should skip its UI when setting the timer. If true, the app should bypass any confirmation UI and simply start the specified timer.

Example intent:

```
public void startTimer(String message, int seconds) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_TIMER)
            .putExtra(AlarmClock.EXTRA_MESSAGE, message)
            .putExtra(AlarmClock.EXTRA_LENGTH, seconds)
            .putExtra(AlarmClock.EXTRA_SKIP_UI, true);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Note:

In order to invoke the [ACTION_SET_TIMER](https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SET_TIMER) intent, your app must have the [SET_ALARM](https://developer.android.com/reference/android/Manifest.permission.html#SET_ALARM) permission:

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SET_TIMER" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Show all alarms

To show the list of alarms, use the [ACTION_SHOW_ALARMS](https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SHOW_ALARMS) action.

Although not many apps will invoke this intent (it's primarily used by system apps), any app that behaves as an alarm clock should implement this intent filter and respond by showing the list of current alarms.

Note: This intent was added in Android 4.4 (API level 19).

Action

`ACTION_SHOW_ALARMS`
(https://developer.android.com/reference/android/provider/AlarmClock.html#ACTION_SHOW_ALARMS)

Data URI

None

MIME Type

None

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SHOW_ALARMS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Calendar

Add a calendar event

To add a new event to the user's calendar, use the `ACTION_INSERT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_INSERT) action and specify the data URI with `Events.CONTENT_URI` (https://developer.android.com/reference/android/provider/CalendarContract.Events.html#CONTENT_URI)
. You can then specify various event details using extras defined below.

Action

`ACTION_INSERT`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_INSERT)

Data URI

`Events.CONTENT_URI`
(https://developer.android.com/reference/android/provider/CalendarContract.Events.html#CONTENT_URI)

MIME Type

`"vnd.android.cursor.dir/event"`

Extras

`EXTRA_EVENT_ALL_DAY`
(https://developer.android.com/reference/android/provider/CalendarContract.html#EXTRA_EVENT_ALL_DAY)

A boolean specifying whether this is an all-day event.

`EXTRA_EVENT_BEGIN_TIME`
(https://developer.android.com/reference/android/provider/CalendarContract.html#EXTRA_EVENT_BEGIN_TIME)

The start time of the event (milliseconds since epoch).

`EXTRA_EVENT_END_TIME`
(https://developer.android.com/reference/android/provider/CalendarContract.html#EXTRA_EVENT_END_TIME)

The end time of the event (milliseconds since epoch).

`TITLE`
(https://developer.android.com/reference/android/provider/CalendarContract.EventsColumns.html#TITLE)

The event title.

`DESCRIPTION`
(https://developer.android.com/reference/android/provider/CalendarContract.EventsColumns.html#DESCRIPTION)

The event description.

`EVENT_LOCATION`
(https://developer.android.com/reference/android/provider/CalendarContract.EventsColumns.html#EVENT_LOCATION)

The event location.

`EXTRA_EMAIL`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_EMAIL)

A comma-separated list of email addresses that specify the invitees.

Many more event details can be specified using the constants defined in the `CalendarContract.EventsColumns` (https://developer.android.com/reference/android/provider/CalendarContract.EventsColumns.html) class.

Example intent:

```
public void addEvent(String title, String location, long begin, long end
    Intent intent = new Intent(Intent.ACTION_INSERT)
            .setData(Events.CONTENT_URI)
            .putExtra(Events.TITLE, title)
            .putExtra(Events.EVENT_LOCATION, location)
            .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, begin)
            .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, end);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.INSERT" />
        <data android:mimeType="vnd.android.cursor.dir/event" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Camera

Capture a picture or video and return it

To open a camera app and receive the resulting photo or video, use the `ACTION_IMAGE_CAPTURE` (https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_IMAGE_CAPTURE) or `ACTION_VIDEO_CAPTURE` (https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_VIDEO_CAPTURE) action. Also specify the URI location where you'd like the camera to save the photo or video,

in the `EXTRA_OUTPUT` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_OUTPUT) extra.

Action

> `ACTION_IMAGE_CAPTURE` (https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_IMAGE_CAPTURE)
>
> or
>
> `ACTION_VIDEO_CAPTURE` (https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_VIDEO_CAPTURE)

Data URI Scheme

> None

MIME Type

> None

Extras

> `EXTRA_OUTPUT` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_OUTPUT)
>
> > The URI location where the camera app should save the photo or video file (as a `Uri` (https://developer.android.com/reference/android/net/Uri.html) object).

When the camera app successfully returns focus to your activity (your app receives the `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) callback), you can access the photo or video at the URI you specified with the `EXTRA_OUTPUT` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_OUTPUT) value.

Note: When you use **`ACTION_IMAGE_CAPTURE`** (https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_IMAGE_CAPTURE) to capture a photo, the camera may also return a downscaled copy (a thumbnail) of the photo in the result **`Intent`** (https://developer.android.com/reference/android/content/Intent.html), saved as a **`Bitmap`** (https://developer.android.com/reference/android/graphics/Bitmap.html) in an extra field named **`"data"`**.

Example intent:

```
static final int REQUEST_IMAGE_CAPTURE = 1;
static final Uri mLocationForPhotos;

public void capturePhoto(String targetFilename) {
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT,
            Uri.withAppendedPath(mLocationForPhotos, targetFilename));
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        // Do other work with full size photo saved in mLocationForPhotos
        ...
    }
}
```

For more information about how to use this intent to capture a photo, including how to create an appropriate `Uri` (https://developer.android.com/reference/android/net/Uri.html) for the output location, read Taking Photos Simply (https://developer.android.com/training/camera/photobasics.html) or Taking Videos Simply (https://developer.android.com/training/camera/videobasics.html).

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

When handling this intent, your activity should check for the `EXTRA_OUTPUT` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_OUTPUT) extra in the incoming `Intent` (https://developer.android.com/reference/android/content/Intent.html), then save the captured image or video at the location specified by that extra and call `setResult()` (https://developer.android.com/reference/android/app/Activity.html#setResult(int, android.content.Intent))

with an `Intent` (https://developer.android.com/reference/android/content/Intent.html) that includes a compressed thumbnail in an extra named "`data`".

Start a camera app in still image mode

To open a camera app in still image mode, use the `INTENT_ACTION_STILL_IMAGE_CAMERA`

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"take a picture"*

(https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_STILL_IMAGE_CAMERA)
action.

Action

`INTENT_ACTION_STILL_IMAGE_CAMERA`
(https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_STILL_IMAGE_CAMERA)

Data URI Scheme

None

MIME Type

None

Extras

None

Example intent:

```
public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.STILL_IMAGE_CAMERA" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Start a camera app in video mode

To open a camera app in video mode, use the `INTENT_ACTION_VIDEO_CAMERA`

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"record a video"*

(https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_VIDEO_CAMERA)
action.

Action

`INTENT_ACTION_VIDEO_CAMERA`
(https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_VIDEO_CAMERA)

Data URI Scheme

None

MIME Type

None

Extras

None

Example intent:

```
public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_VIDEO_CAMERA);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.VIDEO_CAMERA" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Contacts/People App

Select a contact

To have the user select a contact and provide your app access to all the contact information, use the `ACTION_PICK` (https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK) action and specify the MIME type to `Contacts.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html#CONTENT_TYPE)
.

The result `Intent` (https://developer.android.com/reference/android/content/Intent.html) delivered to your `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent))
callback contains the `content:` URI pointing to the selected contact. The response grants your app temporary permissions to read that contact using the Contacts Provider (https://developer.android.com/guide/topics/providers/contacts-provider.html) API even if your app does not include the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission.

Tip: If you need access to only a specific piece of contact information, such as a phone number or email address, instead see the next section about how to select specific contact data (#PickContactData).

Action

`ACTION_PICK`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK)

Data URI Scheme

None

MIME Type

`Contacts.CONTENT_TYPE`
(https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html#CONTENT_TYPE)

Example intent:

```
static final int REQUEST_SELECT_CONTACT = 1;

public void selectContact() {
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(ContactsContract.Contacts.CONTENT_TYPE);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_CONTACT);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    if (requestCode == REQUEST_SELECT_CONTACT && resultCode == RESULT_OK) {
        Uri contactUri = data.getData();
        // Do something with the selected contact at contactUri
        ...
    }
}
```

For information about how to retrieve contact details once you have the contact URI, read Retrieving Details for a Contact (https://developer.android.com/training/contacts-provider/retrieve-details.html). Remember, when you retrieve the contact URI with the above intent, you **do not** need the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission to read details for that contact.

Select specific contact data

To have the user select a specific piece of information from a contact, such as a phone number, email address, or other data type, use the `ACTION_PICK` (https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK) action and specify the MIME type to one of the content types listed below, such as `CommonDataKinds.Phone.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.Phone.html#CONTENT_TYPE) to get the contact's phone number.

If you need to retrieve only one type of data from a contact, this technique with a `CONTENT_TYPE` from the `ContactsContract.CommonDataKinds` (https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.html) classes is more efficient than using the `Contacts.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html#CONTENT_TYPE) (as shown in the previous section) because the result provides you direct access to the desired data without requiring you to perform a more complex query to Contacts Provider (https://developer.android.com/guide/topics/providers/contacts-provider.html).

The result `Intent` (https://developer.android.com/reference/android/content/Intent.html) delivered to your `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) callback contains the `content:` URI pointing to the selected contact data. The response grants your app temporary permissions to read that contact data even if your app does not include the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission.

Action

> `ACTION_PICK` (https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK)

Data URI Scheme

> None

MIME Type

> `CommonDataKinds.Phone.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.Phone.html#CONTENT_TYPE)

Pick from contacts with a phone number.

`CommonDataKinds.Email.CONTENT_TYPE`
(https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.Email.html#CONTENT_TYPE)

Pick from contacts with an email address.

`CommonDataKinds.StructuredPostal.CONTENT_TYPE`
(https://developer.android.com/reference/android/provider/ContactsContract.CommonDataKinds.StructuredPostal.html#CONTENT_TYPE)

Pick from contacts with a postal address.

Or one of many other `CONTENT_TYPE` values under `ContactsContract`
(https://developer.android.com/reference/android/provider/ContactsContract.html).

Example intent:

```
static final int REQUEST_SELECT_PHONE_NUMBER = 1;

public void selectContact() {
    // Start an activity for the user to pick a phone number from contacts
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(CommonDataKinds.Phone.CONTENT_TYPE);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_PHONE_NUMBER);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    if (requestCode == REQUEST_SELECT_PHONE_NUMBER && resultCode == RESULT_OK
        // Get the URI and query the content provider for the phone number
        Uri contactUri = data.getData();
        String[] projection = new String[]{CommonDataKinds.Phone.NUMBER};
        Cursor cursor = getContentResolver().query(contactUri, projection,
                null, null, null);
        // If the cursor returned is valid, get the phone number
        if (cursor != null && cursor.moveToFirst()) {
            int numberIndex = cursor.getColumnIndex(CommonDataKinds.Phone.NUMI
            String number = cursor.getString(numberIndex);
            // Do something with the phone number
            ...
        }
    }
}
```

View a contact

To display the details for a known contact, use the `ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW) action and specify the contact with a `content:` URI as the intent data.

There are primarily two ways to initially retrieve the contact's URI:

- Use the contact URI returned by the `ACTION_PICK` (https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK), shown in the previous section (this approach does not require any app permissions).
- Access the list of all contacts directly, as described in Retrieving a List of Contacts (https://developer.android.com/training/contacts-provider/retrieve-names.html) (this approach requires the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission).

Action

> `ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

Data URI Scheme

> `content:<URI>`

MIME Type

> None. The type is inferred from contact URI.

Example intent:

```
public void viewContact(Uri contactUri) {
    Intent intent = new Intent(Intent.ACTION_VIEW, contactUri);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Edit an existing contact

To edit a known contact, use the `ACTION_EDIT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_EDIT) action, specify

the contact with a `content:` URI as the intent data, and include any known contact information in extras specified by constants in `ContactsContract.Intents.Insert` (https://developer.android.com/reference/android/provider/ContactsContract.Intents.Insert.html).

There are primarily two ways to initially retrieve the contact URI:

- Use the contact URI returned by the `ACTION_PICK` (https://developer.android.com/reference/android/content/Intent.html#ACTION_PICK), shown in the previous section (this approach does not require any app permissions).
- Access the list of all contacts directly, as described in Retrieving a List of Contacts (https://developer.android.com/training/contacts-provider/retrieve-names.html) (this approach requires the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission).

Action

> `ACTION_EDIT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_EDIT)

Data URI Scheme

> `content:<URI>`

MIME Type

> The type is inferred from contact URI.

Extras

> One or more of the extras defined in `ContactsContract.Intents.Insert` (https://developer.android.com/reference/android/provider/ContactsContract.Intents.Insert.html) so you can populate fields of the contact details.

Example intent:

```
public void editContact(Uri contactUri, String email) {
    Intent intent = new Intent(Intent.ACTION_EDIT);
    intent.setData(contactUri);
    intent.putExtra(Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

For more information about how to edit a contact, read Modifying Contacts Using Intents (https://developer.android.com/training/contacts-provider/modify-data.html).

Insert a contact

To insert a new contact, use the `ACTION_INSERT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_INSERT) action, specify `Contacts.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html#CONTENT_TYPE) as the MIME type, and include any known contact information in extras specified by constants in `ContactsContract.Intents.Insert` (https://developer.android.com/reference/android/provider/ContactsContract.Intents.Insert.html).

Action

> `ACTION_INSERT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_INSERT)

Data URI Scheme

> None

MIME Type

> `Contacts.CONTENT_TYPE` (https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html#CONTENT_TYPE)

Extras

> One or more of the extras defined in `ContactsContract.Intents.Insert` (https://developer.android.com/reference/android/provider/ContactsContract.Intents.Insert.html).

Example intent:

```
public void insertContact(String name, String email) {
    Intent intent = new Intent(Intent.ACTION_INSERT);
    intent.setType(Contacts.CONTENT_TYPE);
    intent.putExtra(Intents.Insert.NAME, name);
    intent.putExtra(Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
```

```
    }
}
```

For more information about how to insert a contact, read Modifying Contacts Using Intents (https://developer.android.com/training/contacts-provider/modify-data.html).

Email

Compose an email with optional attachments

To compose an email, use one of the below actions based on whether you'll include attachments, and include email details such as the recipient and subject using the extra keys listed below.

Action

> `ACTION_SENDTO` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SENDTO) (for no attachment) or
> `ACTION_SEND` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) (for one attachment) or
> `ACTION_SEND_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND_MULTIPLE) (for multiple attachments)

Data URI Scheme

> None

MIME Type

> `"text/plain"`
>
> `"*/*"`

Extras

> `Intent.EXTRA_EMAIL` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_EMAIL)
>
> > A string array of all "To" recipient email addresses.

`Intent.EXTRA_CC`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_CC)

A string array of all "CC" recipient email addresses.

`Intent.EXTRA_BCC`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_BCC)

A string array of all "BCC" recipient email addresses.

`Intent.EXTRA_SUBJECT`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_SUBJECT)

A string with the email subject.

`Intent.EXTRA_TEXT`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_TEXT)

A string with the body of the email.

`Intent.EXTRA_STREAM`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_STREAM)

A `Uri` (https://developer.android.com/reference/android/net/Uri.html) pointing to the attachment. If using the `ACTION_SEND_MULTIPLE`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND_MULTIPLE)
action, this should instead be an `ArrayList`
(https://developer.android.com/reference/java/util/ArrayList.html) containing multiple `Uri` (https://developer.android.com/reference/android/net/Uri.html) objects.

Example intent:

```
public void composeEmail(String[] addresses, String subject, Uri attachm
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("*/*");
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

If you want to ensure that your intent is handled only by an email app (and not other text messaging or social apps), then use the **`ACTION_SENDTO`** (https://developer.android.com/reference/android/content/Intent.html#ACTION_SENDTO) action and include the "**`mailto:`**" data scheme. For example:

```
public void composeEmail(String[] addresses, String subject) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:")); // only email apps should handle th
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="*/*" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <data android:scheme="mailto" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

File Storage

Retrieve a specific type of file

To request that the user select a file such as a document or photo and return a reference to your app, use the **`ACTION_GET_CONTENT`** (https://developer.android.com/reference/android/content/Intent.html#ACTION_GET_CONTENT) action and specify your desired MIME type. The file reference returned to your app is transient to your activity's current lifecycle, so if you want to access it later you must import a copy that you can read later. This intent also allows the user to create a new file in the process (for example, instead of selecting an existing photo, the user can capture a new photo with the camera).

The result intent delivered to your `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) method includes data with a URI pointing to the file. The URI could be anything, such as an `http:` URI, `file:` URI, or `content:` URI. However, if you'd like to restrict selectable files to only those that are accessible from a content provider (a `content:` URI) and that are available as a file stream with `openFileDescriptor()` (https://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor(android.net.Uri, java.lang.String)), you should add the `CATEGORY_OPENABLE` (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_OPENABLE) category to your intent.

On Android 4.3 (API level 18) and higher, you can also allow the user to select multiple files by adding `EXTRA_ALLOW_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_ALLOW_MULTIPLE) to the intent, set to `true`. You can then access each of the selected files in a `ClipData` (https://developer.android.com/reference/android/content/ClipData.html) object returned by `getClipData()` (https://developer.android.com/reference/android/content/Intent.html#getClipData()).

Action

`ACTION_GET_CONTENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_GET_CONTENT)

Data URI Scheme

None

MIME Type

The MIME type corresponding to the file type the user should select.

Extras

`EXTRA_ALLOW_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_ALLOW_MULTIPLE)

A boolean declaring whether the user can select more than one file at a time.

`EXTRA_LOCAL_ONLY` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_LOCAL_ONLY)

A boolean that declares whether the returned file must be available directly from the device, rather than requiring a download from a remote service.

Category (optional)

`CATEGORY_OPENABLE`
(https://developer.android.com/reference/android/content/Intent.html#CATEGORY_OPENABLE)

To return only "openable" files that can be represented as a file stream with `openFileDescriptor()`
(https://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor(android.net.Uri, java.lang.String))
.

Example intent to get a photo:

```
static final int REQUEST_IMAGE_GET = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_GET);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    if (requestCode == REQUEST_IMAGE_GET && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        Uri fullPhotoUri = data.getData();
        // Do work with photo saved at fullPhotoUri
        ...
    }
}
```

Example intent filter to return a photo:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.GET_CONTENT" />
        <data android:type="image/*" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- The OPENABLE category declares that the returned file is accessil
             from a content provider that supports OpenableColumns
```
(https://developer.android.com/reference/android/provider/OpenableColumns.html)

```
            and ContentResolver.openFileDescriptor() (https://developer.android.com/r
        <category android:name="android.intent.category.OPENABLE" />
    </intent-filter>
</activity>
```

Open a specific type of file

Instead of retrieving a copy of a file that you must import to your app (by using the `ACTION_GET_CONTENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_GET_CONTENT) action), when running on Android 4.4 or higher, you can instead request to *open* a file that's managed by another app by using the `ACTION_OPEN_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_OPEN_DOCUMENT) action and specifying a MIME type. To also allow the user to instead create a new document that your app can write to, use the `ACTION_CREATE_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CREATE_DOCUMENT) action instead. For example, instead of selecting from existing PDF documents, the `ACTION_CREATE_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CREATE_DOCUMENT) intent allows users to select where they'd like to create a new document (within another app that manages the document's storage)—your app then receives the URI location of where it can write the new document.

Whereas the intent delivered to your `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) method from the `ACTION_GET_CONTENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_GET_CONTENT) action may return a URI of any type, the result intent from `ACTION_OPEN_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_OPEN_DOCUMENT) and `ACTION_CREATE_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CREATE_DOCUMENT) always specify the chosen file as a `content:` URI that's backed by a `DocumentsProvider` (https://developer.android.com/reference/android/provider/DocumentsProvider.html). You can open the file with `openFileDescriptor()` (https://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor(android.net.Uri, java.lang.String)) and query its details using columns from `DocumentsContract.Document` (https://developer.android.com/reference/android/provider/DocumentsContract.Document.html).

The returned URI grants your app long-term read access to the file (also possibly with write access). So the `ACTION_OPEN_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_OPEN_DOCUMENT) action is particularly useful (instead of using `ACTION_GET_CONTENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_GET_CONTENT)) when you want to read an existing file without making a copy into your app, or when you want to open and edit a file in place.

You can also allow the user to select multiple files by adding `EXTRA_ALLOW_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_ALLOW_MULTIPLE) to the intent, set to `true`. If the user selects just one item, then you can retrieve the item from `getData()` (https://developer.android.com/reference/android/content/Intent.html#getData()). If the user selects more than one item, then `getData()` (https://developer.android.com/reference/android/content/Intent.html#getData()) returns null and you must instead retrieve each item from a `ClipData` (https://developer.android.com/reference/android/content/ClipData.html) object that is returned by `getClipData()` (https://developer.android.com/reference/android/content/Intent.html#getClipData()).

Note: Your intent **must** specify a MIME type and **must** declare the `CATEGORY_OPENABLE` (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_OPENABLE) category. If appropriate, you can specify more than one MIME type by adding an array of MIME types with the `EXTRA_MIME_TYPES` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_MIME_TYPES) extra—if you do so, you must set the primary MIME type in `setType()` (https://developer.android.com/reference/android/content/Intent.html#setType(java.lang.String)) to `"*/*"`.

Action

> `ACTION_OPEN_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_OPEN_DOCUMENT)
> or
> `ACTION_CREATE_DOCUMENT` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CREATE_DOCUMENT)

Data URI Scheme

> None

MIME Type

The MIME type corresponding to the file type the user should select.

Extras

`EXTRA_MIME_TYPES`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_MIME_TYPES)

An array of MIME types corresponding to the types of files your app is requesting. When you use this extra, you must set the primary MIME type in `setType()`
(https://developer.android.com/reference/android/content/Intent.html#setType(java.lang.String))
to `"*/*"`.

`EXTRA_ALLOW_MULTIPLE`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_ALLOW_MULTIPLE)

A boolean that declares whether the user can select more than one file at a time.

`EXTRA_TITLE`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_TITLE)

For use with `ACTION_CREATE_DOCUMENT`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_CREATE_DOCUMENT)
to specify an initial file name.

`EXTRA_LOCAL_ONLY`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_LOCAL_ONLY)

A boolean that declares whether the returned file must be available directly from the device, rather than requiring a download from a remote service.

Category

`CATEGORY_OPENABLE`
(https://developer.android.com/reference/android/content/Intent.html#CATEGORY_OPENABLE)

To return only "openable" files that can be represented as a file stream with `openFileDescriptor()`

(https://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor(android.net.Uri, java.lang.String))

.

Example intent to get a photo:

```
static final int REQUEST_IMAGE_OPEN = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
    intent.setType("image/*");
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    // Only the system receives the ACTION_OPEN_DOCUMENT, so no need to test.
    startActivityForResult(intent, REQUEST_IMAGE_OPEN);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    if (requestCode == REQUEST_IMAGE_OPEN && resultCode == RESULT_OK) {
        Uri fullPhotoUri = data.getData();
        // Do work with full size photo saved at fullPhotoUri
        ...
    }
}
```

Third party apps cannot actually respond to an intent with the **`ACTION_OPEN_DOCUMENT`** (https://developer.android.com/reference/android/content/Intent.html#ACTION_OPEN_DOCUMENT) action. Instead, the system receives this intent and displays all the files available from various apps in a unified user interface.

To provide your app's files in this UI and allow other apps to open them, you must implement a **`DocumentsProvider`** (https://developer.android.com/reference/android/provider/DocumentsProvider.html) and include an intent filter for **`PROVIDER_INTERFACE`** (https://developer.android.com/reference/android/provider/DocumentsContract.html#PROVIDER_INTERFACE) ("**`android.content.action.DOCUMENTS_PROVIDER`**"). For example:

```
<provider ...
    android:grantUriPermissions="true"
    android:exported="true"
    android:permission="android.permission.MANAGE_DOCUMENTS">
    <intent-filter>
        <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
```

```
    </intent-filter>
</provider>
```

For more information about how to make the files managed by your app openable from other apps, read the Storage Access Framework (https://developer.android.com/guide/topics/providers/document-provider.html) guide.

Local Actions

Call a car

To call a taxi, use the `ACTION_RESERVE_TAXI_RESERVATION` (https://developer.android.com/reference/com/google/android/gms/actions/ReserveIntents.html#ACTION_RESERVE_TAXI_RESERVATION) action.

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"get me a taxi"*
- *"call me a car"*

(Wear OS only)

Note: Apps must ask for confirmation from the user before completing the action.

Action

`ACTION_RESERVE_TAXI_RESERVATION` (https://developer.android.com/reference/com/google/android/gms/actions/ReserveIntents.html#ACTION_RESERVE_TAXI_RESERVATION)

Data URI

None

MIME Type

None

Extras

None

Example intent:

```
public void callCar() {
    Intent intent = new Intent(ReserveIntents.ACTION_RESERVE_TAXI_RESERVATION
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="com.google.android.gms.actions.RESERVE_TAXI_RESI
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Maps

Show a location on a map

To open a map, use the `ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW) action and specify the location information in the intent data with one of the schemes defined below.

Action

`ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

Data URI Scheme

`geo:`*`latitude`*`,`*`longitude`*

Show the map at the given longitude and latitude.
Example: "`geo:47.6,-122.3`"

`geo:`*`latitude`*`,`*`longitude`*`?z=`*`zoom`*

Show the map at the given longitude and latitude at a certain zoom level. A zoom level of 1 shows the whole Earth, centered at the given *lat*,*lng*. The highest (closest) zoom level is 23.

Example: "`geo:47.6,-122.3?z=11`"

`geo:0,0?q=lat,lng(label)`

Show the map at the given longitude and latitude with a string label.
Example: "`geo:0,0?q=34.99,-106.61(Treasure)`"

`geo:0,0?q=my+street+address`

Show the location for "my street address" (may be a specific address or location query).
Example: "`geo:0,0?q=1600+Amphitheatre+Parkway%2C+CA`"

★ **Note:** All strings passed in the `geo` URI must be encoded. For example, the string `1st & Pike, Seattle` should become `1st%20%26%20Pike%2C%20Seattle`. Spaces in the string can be encoded with `%20` or replaced with the plus sign (`+`).

MIME Type

None

Example intent:

```
public void showMap(Uri geoLocation) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(geoLocation);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <data android:scheme="geo" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Music or Video

Play a media file

To play a music file, use the `ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW) action and specify the URI location of the file in the intent data.

Action

> `ACTION_VIEW` (https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

Data URI Scheme

> `file:<URI>`
>
> `content:<URI>`
>
> `http:<URL>`

MIME Type

> `"audio/*"`
>
> `"application/ogg"`
>
> `"application/x-ogg"`
>
> `"application/itunes"`
>
> **Or any other that your app may require.**

Example intent:

```
public void playMedia(Uri file) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(file);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <data android:type="audio/*" />
        <data android:type="application/ogg" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Play music based on a search query

To play music based on a search query, use the `INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH` (https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH) intent. An app may fire this intent in response to the user's voice command to play music. The receiving app for this intent performs a search within its inventory to match existing content to the given query and starts playing that content.

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"play michael jackson billie jean"*

This intent should include the `EXTRA_MEDIA_FOCUS` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_FOCUS) string extra, which specifies the inteded search mode. For example, the search mode can specify whether the search is for an artist name or song name.

Action

> `INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH` (https://developer.android.com/reference/android/provider/MediaStore.html#INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH)

Data URI Scheme

> None

MIME Type

None

Extras

`MediaStore.EXTRA_MEDIA_FOCUS`
(https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_FOCUS)
(required)

Indicates the search mode (whether the user is looking for a particular artist, album, song, or playlist). Most search modes take additional extras. For example, if the user is interested in listening to a particular song, the intent might have three additional extras: the song title, the artist, and the album. This intent supports the following search modes for each value of
`EXTRA_MEDIA_FOCUS`
(https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_FOCUS)
:

***Any* - `"vnd.android.cursor.item/*"`**

Play any music. The receiving app should play some music based on a smart choice, such as the last playlist the user listened to.

Additional extras:

- `QUERY`
  (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - An empty string. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

***Unstructured* - `"vnd.android.cursor.item/*"`**

Play a particular song, album or genre from an unstructured search query. Apps may generate an intent with this search mode when they can't identify the type of content the user wants to listen to. Apps should use more specific search modes when possible.

Additional extras:

- `QUERY`
  (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)

(required) - A string that contains any combination of: the artist, the album, the song name, or the genre.

***Genre* - `Audio.Genres.ENTRY_CONTENT_TYPE`** (https://developer.android.com/reference/android/provider/MediaStore.Audio.Genres.html#ENTRY_CONTENT_TYPE)

Play music of a particular genre.

Additional extras:

- "`android.intent.extra.genre`" (required) - The genre.
- `QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - The genre. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

***Artist* - `Audio.Artists.ENTRY_CONTENT_TYPE`** (https://developer.android.com/reference/android/provider/MediaStore.Audio.Artists.html#ENTRY_CONTENT_TYPE)

Play music from a particular artist.

Additional extras:

- `EXTRA_MEDIA_ARTIST` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ARTIST)
  (required) - The artist.
- "`android.intent.extra.genre`" - The genre.
- `QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - A string that contains any combination of the artist or the genre. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

***Album* - `Audio.Albums.ENTRY_CONTENT_TYPE`** (https://developer.android.com/reference/android/provider/MediaStore.Audio.Albums.html#ENTRY_CONTENT_TYPE)

Play music from a particular album.

Additional extras:

- EXTRA_MEDIA_ALBUM (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ALBUM)
  (required) - The album.
- EXTRA_MEDIA_ARTIST (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ARTIST)
  - The artist.
- "`android.intent.extra.genre`" - The genre.
- QUERY (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - A string that contains any combination of the album or the artist. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

***Song* -** "`vnd.android.cursor.item/audio`"

Play a particular song.

Additional extras:

- EXTRA_MEDIA_ALBUM (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ALBUM)
  - The album.
- EXTRA_MEDIA_ARTIST (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ARTIST)
  - The artist.
- "`android.intent.extra.genre`" - The genre.
- EXTRA_MEDIA_TITLE (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_TITLE)
  (required) - The song name.

- `QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - A string that contains any combination of: the album, the artist, the genre, or the title. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

***Playlist* - `Audio.Playlists.ENTRY_CONTENT_TYPE`** (https://developer.android.com/reference/android/provider/MediaStore.Audio.Playlists.html#ENTRY_CONTENT_TYPE)

Play a particular playlist or a playlist that matches some criteria specified by additional extras.

Additional extras:

- `EXTRA_MEDIA_ALBUM` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ALBUM)
  - The album.
- `EXTRA_MEDIA_ARTIST` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_ARTIST)
  - The artist.
- "`android.intent.extra.genre`" - The genre.
- "`android.intent.extra.playlist`" - The playlist.
- `EXTRA_MEDIA_TITLE` (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_TITLE)
  - The song name that the playlist is based on.
- `QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)
  (required) - A string that contains any combination of: the album, the artist, the genre, the playlist, or the title. This extra is always provided for backward compatibility: existing apps that do not know about search modes can process this intent as an unstructured search.

Example intent:

If the user wants to listen to music from a particular artist, a search app may generate the following intent:

```
public void playSearchArtist(String artist) {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH
    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
                    MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_ARTIST, artist);
    intent.putExtra(SearchManager.QUERY, artist);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.MEDIA_PLAY_FROM_SEARCH" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

When handling this intent, your activity should check the value of the **`EXTRA_MEDIA_FOCUS`** (https://developer.android.com/reference/android/provider/MediaStore.html#EXTRA_MEDIA_FOCUS) extra in the incoming **`Intent`** (https://developer.android.com/reference/android/content/Intent.html) to determine the search mode. Once your activity has identified the search mode, it should read the values of the additional extras for that particular search mode. With this information your app can then perform the search within its inventory to play the content that matches the search query. For example:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    Intent intent = this.getIntent();
    if (intent.getAction().compareTo(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_

        String mediaFocus = intent.getStringExtra(MediaStore.EXTRA_MEDIA_FOCUS
        String query = intent.getStringExtra(SearchManager.QUERY);

        // Some of these extras may not be available depending on the search m
        String album = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ALBUM);
        String artist = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ARTIST);
        String genre = intent.getStringExtra("android.intent.extra.genre");
        String playlist = intent.getStringExtra("android.intent.extra.playlist
        String title = intent.getStringExtra(MediaStore.EXTRA_MEDIA_TITLE);
```

```
        // Determine the search mode and use the corresponding extras
        if (mediaFocus == null) {
            // 'Unstructured' search mode (backward compatible)
            playUnstructuredSearch(query);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/*") == 0) {
            if (query.isEmpty()) {
                // 'Any' search mode
                playResumeLastPlaylist();
            } else {
                // 'Unstructured' search mode
                playUnstructuredSearch(query);
            }

        } else if (mediaFocus.compareTo(MediaStore.Audio.Genres.ENTRY_CONTENT
            // 'Genre' search mode
            playGenre(genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Artists.ENTRY_CONTEN
            // 'Artist' search mode
            playArtist(artist, genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Albums.ENTRY_CONTENT
            // 'Album' search mode
            playAlbum(album, artist);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/audio") == 0
            // 'Song' search mode
            playSong(album, artist, genre, title);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Playlists.ENTRY_CONTI
            // 'Playlist' search mode
            playPlaylist(album, artist, genre, playlist, title);
        }
    }
}
```

New Note

Create a note

To create a new note, use the **`ACTION_CREATE_NOTE`** (https://developers.google.com/android/reference/com/google/android/gms/actions/NoteIntents#ACTION_CREATE_NOTE)

action and specify note details such as the subject and text using extras defined below.

Note: Apps must ask for confirmation from the user before completing the action.

Action

ACTION_CREATE_NOTE
(https://developers.google.com/android/reference/com/google/android/gms/actions/NoteIntents#ACTION_CREATE_NOTE)

Data URI Scheme

None

MIME Type

PLAIN_TEXT_TYPE
(https://developer.android.com/reference/org/apache/http/protocol/HTTP.html#PLAIN_TEXT_TYPE)

"*/*"

Extras

EXTRA_NAME
(https://developers.google.com/android/reference/com/google/android/gms/actions/NoteIntents#EXTRA_NAME)

A string indicating the title or subject of the note.

EXTRA_TEXT
(https://developers.google.com/android/reference/com/google/android/gms/actions/NoteIntents#EXTRA_TEXT)

A string indicating the text of the note.

Example intent:

```
public void createNote(String subject, String text) {
    Intent intent = new Intent(NoteIntents.ACTION_CREATE_NOTE)
            .putExtra(NoteIntents.EXTRA_NAME, subject)
            .putExtra(NoteIntents.EXTRA_TEXT, text);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="com.google.android.gms.actions.CREATE_NOTE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="*/*" />
    </intent-filter>
</activity>
```

Phone

Initiate a phone call

To open the phone app and dial a phone number, use the `ACTION_DIAL` (https://developer.android.com/reference/android/content/Intent.html#ACTION_DIAL) action and specify a phone number using the URI scheme defined below. When the phone app opens, it displays the phone number but the user must press the *Call* button to begin the phone call.

To place a phone call directly, use the `ACTION_CALL`

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"call 555-5555"*
- *"call bob"*
- *"call voicemail"*

(https://developer.android.com/reference/android/content/Intent.html#ACTION_CALL) action and specify a phone number using the URI scheme defined below. When the phone app opens, it begins the phone call; the user does not need to press the *Call* button.

The `ACTION_CALL` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CALL) action requires that you add the `CALL_PHONE` permission to your manifest file:

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

Action

- `ACTION_DIAL` (https://developer.android.com/reference/android/content/Intent.html#ACTION_DIAL) - Opens the dialer or phone app.
- `ACTION_CALL` (https://developer.android.com/reference/android/content/Intent.html#ACTION_CALL) - Places a phone call (requires the `CALL_PHONE` permission)

Data URI Scheme

- `tel:<phone-number>`
- `voicemail:<phone-number>`

MIME Type

None

Valid telephone numbers are those defined in the IETF RFC 3966 (http://tools.ietf.org/html/rfc3966). Valid examples include the following:

- `tel:2125551212`
- `tel:(212) 555 1212`

The Phone's dialer is good at normalizing schemes, such as telephone numbers. So the scheme described isn't strictly required in the `Uri.parse()` (https://developer.android.com/reference/android/net/Uri.html#parse(java.lang.String)) method. However, if you have not tried a scheme or are unsure whether it can be handled, use the `Uri.fromParts()` (https://developer.android.com/reference/android/net/Uri.html#fromParts(java.lang.String, java.lang.String, java.lang.String)) method instead.

Example intent:

```
public void dialPhoneNumber(String phoneNumber) {
    Intent intent = new Intent(Intent.ACTION_DIAL);
    intent.setData(Uri.parse("tel:" + phoneNumber));
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Search

Search using a specific app

To support search within the context of your app, declare an intent filter in your app with the `SEARCH_ACTION` action, as shown in the example intent filter below.

(https://developers.google.c
actions/system/#system_act

Google Voice Actions

- *"search for cat videos on myvideoapp"*

Action

`"com.google.android.gms.actions.SEARCH_ACTION"`

Support search queries from Google Voice Actions.

Extras

`QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY)

A string that contains the search query.

Example intent filter:

```
<activity android:name=".SearchActivity">
    <intent-filter>
        <action android:name="com.google.android.gms.actions.SEARCH_ACTION"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

Perform a web search

To initiate a web search, use the `ACTION_WEB_SEARCH` (https://developer.android.com/reference/android/content/Intent.html#ACTION_WEB_SEARCH) action and specify the search string in the `SearchManager.QUERY` (https://developer.android.com/reference/android/app/SearchManager.html#QUERY) extra.

Action

`ACTION_WEB_SEARCH`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_WEB_SEARCH)

Data URI Scheme

None

MIME Type

None

Extras

`SearchManager.QUERY`
(https://developer.android.com/reference/android/app/SearchManager.html#QUERY)

The search string.

Example intent:

```
public void searchWeb(String query) {
    Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
    intent.putExtra(SearchManager.QUERY, query);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Settings

Open a specific section of Settings

To open a screen in the system settings when your app requires the user to change something, use one of the following intent actions to open the settings screen respective to the action name.

Action

`ACTION_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_SETTINGS)
`ACTION_WIRELESS_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_WIRELESS_SETTINGS)
`ACTION_AIRPLANE_MODE_SETTINGS`

(https://developer.android.com/reference/android/provider/Settings.html#ACTION_AIRPLANE_MODE_SETTINGS)

`ACTION_WIFI_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_WIFI_SETTINGS)

`ACTION_APN_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_APN_SETTINGS)

`ACTION_BLUETOOTH_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_BLUETOOTH_SETTINGS)

`ACTION_DATE_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_DATE_SETTINGS)

`ACTION_LOCALE_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_LOCALE_SETTINGS)

`ACTION_INPUT_METHOD_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_INPUT_METHOD_SETTINGS)

`ACTION_DISPLAY_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_DISPLAY_SETTINGS)

`ACTION_SECURITY_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_SECURITY_SETTINGS)

`ACTION_LOCATION_SOURCE_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_LOCATION_SOURCE_SETTINGS)

`ACTION_INTERNAL_STORAGE_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_INTERNAL_STORAGE_SETTINGS)

`ACTION_MEMORY_CARD_SETTINGS`
(https://developer.android.com/reference/android/provider/Settings.html#ACTION_MEMORY_CARD_SETTINGS)

See the `Settings` (https://developer.android.com/reference/android/provider/Settings.html) documentation for additional settings screens that are available.

Data URI Scheme

None

MIME Type

None

Example intent:

```
public void openWifiSettings() {
    Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Text Messaging

Compose an SMS/MMS message with attachment

To initiate an SMS or MMS text message, use one of the intent actions below and specify message details such as the phone number, subject, and message body using the extra keys listed below.

Action

`ACTION_SENDTO` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SENDTO) or `ACTION_SEND` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) or `ACTION_SEND_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND_MULTIPLE)

Data URI Scheme

`sms:<phone_number>`

`smsto:<phone_number>`

`mms:<phone_number>`

`mmsto:<phone_number>`

Each of these schemes are handled the same.

MIME Type

`"text/plain"`

`"image/*"`

`"video/*"`

Extras

`"subject"`

A string for the message subject (usually for MMS only).

`"sms_body"`

A string for the text message.

`EXTRA_STREAM`
(https://developer.android.com/reference/android/content/Intent.html#EXTRA_STREAM)

A `Uri` (https://developer.android.com/reference/android/net/Uri.html) pointing to the image or video to attach. If using the `ACTION_SEND_MULTIPLE` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND_MULTIPLE) action, this extra should be an `ArrayList` (https://developer.android.com/reference/java/util/ArrayList.html) of `Uri` (https://developer.android.com/reference/android/net/Uri.html)s pointing to the images/videos to attach.

Example intent:

```
public void composeMmsMessage(String message, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setType(HTTP.PLAIN_TEXT_TYPE);
    intent.putExtra("sms_body", message);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

If you want to ensure that your intent is handled only by a text messaging app (and not other email or social apps), then use the `ACTION_SENDTO` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SENDTO) action and include the `"smsto:"` data scheme. For example:

```
public void composeMmsMessage(String message, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setData(Uri.parse("smsto:"));  // This ensures only SMS apps respo
```

```
    intent.putExtra("sms_body", message);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

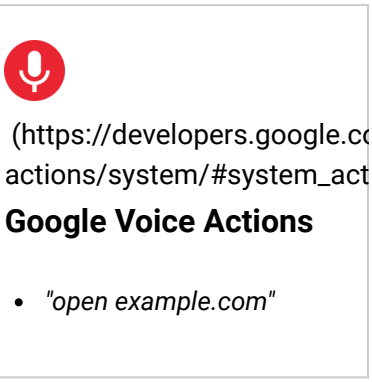
```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="text/plain" />
        <data android:type="image/*" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Note: If you're developing an SMS/MMS messaging app, you must implement intent filters for several additional actions in order to be available as the *default SMS app* on Android 4.4 and higher. For more information, see the documentation at `Telephony` (https://developer.android.com/reference/android/provider/Telephony.html).

Web Browser

Load a web URL

To open a web page, use the `ACTION_VIEW`



(https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW) action and specify the web URL in the intent data.

Action

`ACTION_VIEW`
(https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

Data URI Scheme

`http:<URL>`
`https:<URL>`

MIME Type

`"text/plain"`

`"text/html"`

`"application/xhtml+xml"`

`"application/vnd.wap.xhtml+xml"`

Example intent:

```
public void openWebPage(String url) {
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Example intent filter:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <!-- Include the host attribute if you want your app to respond
             only to URLs with your app's domain. -->
        <data android:scheme="http" android:host="www.example.com" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- The BROWSABLE category is required to get links from web pages.
        <category android:name="android.intent.category.BROWSABLE" />
    </intent-filter>
</activity>
```

Tip: If your Android app provides functionality similar to your web site, include an intent filter for URLs that point to your web site. Then, if users have your app installed, links from emails or other web pages pointing to your web site open your Android app instead of your web page.

Verify Intents with the Android Debug Bridge

To verify that your app responds to the intents that you want to support, you can use the **adb** (https://developer.android.com/tools/help/adb.html) tool to fire specific intents:

1. Set up an Android device for development (https://developer.android.com/tools/device.html#setting-up), or use a virtual device (https://developer.android.com/tools/devices/emulator.html#avds).
2. Install a version of your app that handles the intents you want to support.
3. Fire an intent using **adb**:

   ```
   adb shell am start -a <ACTION> -t <MIME_TYPE> -d <DATA> \
     -e <EXTRA_NAME> <EXTRA_VALUE> -n <ACTIVITY>
   ```

   For example:

   ```
   adb shell am start -a android.intent.action.DIAL \
     -d tel:555-5555 -n org.example.MyApp/.MyActivity
   ```

4. If you defined the required intent filters, your app should handle the intent.

For more information, see ADB Shell Commands (https://developer.android.com/tools/help/shell.html#am).

Last updated July 2, 2018.

Twitter
Follow @AndroidDev on Twitter

Google+
Follow Android Developers on Google+

YouTube
Check out Android Developers on YouTube