Launching External Apps

See also

Launching ODK Collect from External Apps

Launching external apps to populate single fields

ODK Collect can launch external applications using the `ex:intentAction(intentExtras)` appearance on string, integer, decimal, image, video, audio and file fields. The application launched could be one that the enumerator uses and quits without Collect needing any data from it. For example, a form could configure the Maps.me application to provide directions to a destination and not need any information back from the app. An external application could also be used to populate the field that launched it. In order to populate a field, the app that is launched must be designed to return a value as described in the external app design section below.

This feature configures an Android intent. Intents are messaging objects used to request an action from another app component. Learn more in the Android docs.

Collect builds the action using the text immediately after `ex:` and before an opening parenthesis (if it exists). For example, in the string `ex:org.opendatakit.counter(intentExtras)`, the action name is `org.opendatakit.counter`. An action name must include a namespace, such as `org.opendatakit` or `com.google`. For example, `ex:org.opendatakit.counter` is valid but `ex:counter` is not.

The parameters defined in the optional parentheses represent extended data ("extras") to be added to the intent. Extras are specified by a comma-delimited list of `name=value` pairs. The text to the left of the equals sign represents the extra name and may require a namespace. The text to the right of the equals sign represents the extra value. All extra values are sent as strings.

The values of extras can be:

- XPath expressions referring to other fields and including function calls
- String literals defined in single quotes
- Raw integers or decimals

An extra named `value` that holds the current value for the current field is always passed with the intent. Additional parameters with user-defined names can also be specified. There are two reserved names: `value` and `uri_data`.

Since Collect v1.16.0, the data that the intent operates on can be set by using the reserved uri_data parameter. This is particularly useful for implicit intents such as `android.intent.action.SENDTO`.

XLSForm

survey

type	name	label	appearance
integer	counter	Click launch to start the counter app	ex:org.opendatakit.counter(form_id='counter-form', form_name='Counter Form', question_id='1', question_name='Counter')

In the examples above, the extras specified have names `form_id`, `form_name`, `question_id`, and `question_name`.

Specifying a URI as intent data

Since Collect v1.16.0, the value for the reserved parameter name `uri_data` is converted to a URI and used as the data for the intent. The intent data determines which application to launch when using implicit intents such as SENDTO. For example:

`ex:android.intent.action.SENDTO(uri_data='smsto:5555555', sms_body=${message})`
: Launches a new message in an SMS app with the destination number set to `5555555` and the message body set to the contents of the `message` field.

`ex:android.intent.action.SENDTO(uri_data='mailto:example@example.com?subject=${subject}&body=${message})`
: Launches a new message in an email app with destination address set to `example@example.com`, the subject set to the contents of the `subject` field and the body set to the contents of the `message` field.

`ex:android.intent.action.DIAL(uri_data='tel:5555555')`
: Launches a phone dialer with the number `5555555` as the number to dial.

Notice that the URI must include a scheme, such as `mailto:` or `https://`.

Designing an app to return a single value to Collect

When an activity that is launched from a string, integer or decimal question returns to Collect, Collect will look for an intent extra named `value` and use its value to populate the field that triggered the application launch. Your app should provide a `String` extra named `value` and set its value to what you want to send to Collect. See a counter app for an example of how this is done.

When an activity that is launched from an image, video, audio or file question returns to Collect, Collect will take the contents of the ClipData associated with the return `Intent` and save it to a file. Any intent extras are ignored. To set your app's return `Intent` `ClipData`:

```
Uri uri = ...
Intent returnIntent = new Intent();
returnIntent.clipData = ClipData.newRawUri(null, uri);
intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
setResult(Activity.RESULT_OK, intent);
finish();
```

If using a file question, all file types will be accepted. If you know that your custom app will return an image, video or audio file, use the more specific question type so that Collect will show a type-appropriate preview of the returned file.

Additionally, your external app can receive values from Collect through `Intent` extras. As described above, Collect will always provide an extra with name `value` and the current value of the field. You can also document additional extras that your app uses for form designers to specify. There are two reserved names: `value` and `uri_data`. All extras sent to your app from Collect will be of type `String`. You must document any restrictions on the extra values and validate them on app launch. For example, you might document that a `test_mode` extra accepts values `yes` or `no`. Collect passes on any text it is given as extra values without validation so your app should define fallback behavior in case it is given an invalid value.

External apps to populate multiple fields

A `field-list` group can have an `intent` attribute that allows an external application to populate it. Notice that the `ex:` prefix used when populating a single field is not included to populate multiple fields.

XLSForm

survey

type	name	label	appearance	body::intent
begin_group	mygroup	Fields to populate	field-list	org.mycompany.myapp(my_text='Some text', uuid=/myform/meta/instanceID)
text	some_text	Some text		
integer	some_integer	Some integer		
decimal	some_decimal	Some decimal		
image	some_image	Some image		
video	some_video	Some video		
audio	some_audio	Some audio		
file	some_file	Some file		
end_group				

```
<group ref="/myform/mygroup" appearance="field-list"
       intent="org.mycompany.myapp(my_text='Some text',
                                    uuid=/myform/meta/instanceID)">
  <label>Fields to populate</label>
  <input ref="/myform/mygroup/some_text">
    <label>Some text</label>
  </input>
  <input ref="/myform/mygroup/some_integer">
    <label>Some integer</label>
  </input>
  <input ref="/myform/mygroup/some_decimal">
    <label>Some decimal</label>
  </input>
  <input ref="/myform/mygroup/some_image">
    <label>Some image</label>
  </input>
  <input ref="/myform/mygroup/some_video">
    <label>Some video</label>
  </input>
  <input ref="/myform/mygroup/some_audio">
    <label>Some audio</label>
  </input>
  <input ref="/myform/mygroup/some_file">
    <label>Some file</label>
  </input>
</group>
```

The `intent` attribute is only used when the group has an `appearance` of `field-list`. The format and the functionality of the `intent` value is the same as above. If the `Intent` extras returned by the external application contains values with keys that match the type and the name of the sub-fields, then the values from the `Intent` extras overwrite the current values of those sub-fields.

The external app is launched with the parameters that are defined in the intent string plus the values of all the sub-fields that are either text, decimal, integer, or binary (filename is sent). Any other sub-field is invisible to the external app.

Since Collect v1.30.0, it is possible to populate binary fields (image, video, audio or file) in field lists. An app that returns one or more binary files to Collect as part of a field list must provide a content URI as the value for each `extra` that corresponds to a binary question to populate. Additionally, the external application must specify ClipData items for each URI it returns. Your app can then grant Collect temporary permissions to the files using the Intent.FLAG_GRANT_READ_URI_PERMISSION intent flag. See a similar code example above.

Typically, an external app creator decides on the names of input and output extras and documents those. Form designers use the names of the expected input extras in the `appearance` of the `field-list` used to launch the external app (e.g. `my_text` in `org.mycompany.myapp(my_text='Some text')` above). Form designers use the names of the expected outputs from the external app to name the questions in the field list.