ODK XForm Specification

Work in Progress - Pending peer-review and approval

Introduction

The ODK XForm specification is a subset of the far larger XForm 1.0 specification. It contains a few additional features not found in the XForm specification.

The ODK XForm Specification continues to evolve. On the web you will often see references to *JavaRosa* and *OpenRosa* when the spec is discussed. OpenRosa was the name of the initial specification that formed the basis of the ODK specification. JavaRosa is the name of a Java library that implements the OpenRosa specification. ODK, JavaRosa, OpenRosa are still often used to refer to the same, but we recommend using this document to build tools that are compliant with the ODK Ecosystem.

The document assumes at least a fair understanding of XML and XPath. It is also useful to refer to XForms 1.0 for details about shared features.

Structure

The high-level form definition is structured as follows:

- model
  - instance
  - bindings
- body

The model contains the **instance**(s) and the **bindings**. The first instance is the XML data structure of the *record* that is captured with the form. A binding describes an individual instance node and includes information such as *datatype, skip logic, calculations, and more*.

The **body** contains the information required to *display* a form.

Below is an example of a complete and valid ODK XForm:

```
<?xml version="1.0"?>
<h:html xmlns="http://www.w3.org/2002/xforms"
        xmlns:ev="http://www.w3.org/2001/xml-events"
        xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:jr="http://openrosa.org/javarosa"
        xmlns:orx="http://openrosa.org/xforms/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
    <h:head>
        <h:title>My Survey</h:title>
        <model>
            <instance>
                <data id="mysurvey" version="2014083101">
                    <firstname></firstname>
                    <lastname></lastname>
                    <age></age>
                    <meta>
                        <instanceID/>
                    </meta>
                </data>
            </instance>
            <bind nodeset="/data/firstname" type="string" requi
            <bind nodeset="/data/lastname"  type="string" />
            <bind nodeset="/data/age" type="integer" />
        </model>
    </h:head>
    <h:body>
        <input ref="/data/firstname">
          <label>What is your first name?</label>
        </input>
        <input ref="/data/lastname">
          <label>What is your last name?</label>
        </input>
        <input ref="/data/age">
          <label>What is your age?</label>
        </input>
    </h:body>
</h:html>
```

Instance

A `<model>` can have multiple instances as childnodes. The first and required `<instance>` is called the *primary instance* and represents the data structure of the record that will be created and submitted with the form. Additional instances are called *secondary instances*.

Primary Instance

The primary instance should contain a single childnode. In the example below `<household>` will be populated with data and submitted. The primary instance's single child is the **document root** that XPath expressions are evaluated on (e.g. in the instance below the value of `/household/person/age` is 10).

```
<instance>
```

```
    <household id="mysurvey" version="2014083101">
        <person>
            <firstname/>
            <lastname/>
            <age>10</age>
        </person>
        <meta>
          <instanceID/>
        </meta>
    </household>
</instance>
```

Any value inside a primary instance is considered a default value for that question. If that node has a corresponding input element that value will be displayed to the user when the question is parsed.

Nodes inside a primary instance can contain attributes. The client application normally retains the attribute when a record is submitted. There are 3 pre-defined instance attributes:

attribute	description
`id`	on the childnode of the primary instance: This is the unique ID at which the form is identified the server that publishes the Form and receives data submissions. For more information see this Form List Specification.
`version`	on the childnode of the primary instance can contain any string value.
`jr:template`	on any repeat group node: This serves to define a default template for repeats and is useful if any of the leaf nodes inside a repeat contains a default value. It is not transmitted in the record. For more details, see the repeats section.

The primary instance also includes a special type of nodes for metadata inside the `<meta>` block. pending - See the Metadata section

Secondary Instances

Secondary instances are used to pre-load data inside a form. This data is searchable in XPath. At the moment the only use case is so-called *cascading selections* where the available options of a multiple-choice question can be filtered based on an earlier answer.

A secondary instance should get a unique `id` attribute on the `<instance>` node. This allows apps to query the data (which is outside the root, ie. the primary instance, and would normally not be reachable). It uses the the `instance('cities')/root/item[country='nl']` syntax to do this.

```
<instance>
    <household id="mysurvey" version="2014083101">
        <person>
            <firstname/>
            <lastname/>
            <age>10</age>
        </person>
        <meta>
          <instanceID/>
        </meta>
    </household>
</instance>
<instance id="cities">
    <root>
        <item>
            <itextId>static_instance-cities-0</itextId>
            <country>nl</country>
            <name>ams</name>
        </item>
        <item>
            <itextId>static_instance-cities-1</itextId>
            <country>usa</country>
            <name>den</name>
      </item>
      <item>
            <itextId>static_instance-cities-2</itextId>
            <country>usa</country>
            <name>nyc</name>
      </item>
      <item>
        <itextId>static_instance-cities-5</itextId>
        <country>nl</country>
        <name>dro</name>
      </item>
    </root>
</instance>
<instance id="neighborhoods">
    <root>
        <item>
            <itextId>static_instance-neighborhoods-0</itextId>
            <city>nyc</city>
            <country>usa</country>
            <name>bronx</name>
        </item>
        <item>
            <itextId>static_instance-neighborhoods-3</itextId>
            <city>ams</city>
            <country>nl</country>
```

```
            <name>wes</name>
        </item>
        <item>
            <itextId>static_instance-neighborhoods-4</itextId>
            <city>den</city>
            <country>usa</country>
            <name>goldentriangle</name>
        </item>
        <item>
            <itextId>static_instance-neighborhoods-8</itextId>
            <city>dro</city>
            <country>nl</country>
            <name>haven</name>
        </item>
    </root>
</instance>
```

Bindings

A `<bind>` element wires together a primary instance node and the presentation of the corresponding question to the user. It is used to describe the datatype and various kinds of logic related to the data. A bind can refer to any node in the primary instance including repeated nodes_. It may or may not have a corresponding presentation node in the body.

```
<bind nodeset="/d/my_intro" type="string" readonly="true()"/>
<bind nodeset="/d/text_widgets/my_string" type="string"/>
<bind nodeset="/d/text_widgets/my_long_text" type="string"/>
<bind nodeset="/d/number_widgets/my_int" type="int" constraint=
<bind nodeset="/d/number_widgets/my_decimal" type="decimal" con
<bind nodeset="/d/dt/my_date" type="date" constraint=". &gt;= t
<bind nodeset="/d/dt/my_time" type="time"/>
<bind nodeset="/d/dt/dateTime" type="dateTime"/>
<bind nodeset="/d/s/my_select" type="select" constraint="select
<bind nodeset="/d/s/my_select1" type="select1"/>
<bind nodeset="/d/geo/my_geopoint" type="geopoint"/>
<bind nodeset="/d/geo/my_geotrace" type="geotrace"/>
<bind nodeset="/d/geo/my_geoshape" type="geoshape"/>
<bind nodeset="/d/media/my_image" type="binary"/>
<bind nodeset="/d/media/my_audio" type="binary"/>
<bind nodeset="/d/media/my_video" type="binary"/>
<bind nodeset="/d/media/my_barcode" type="barcode"/>
<bind nodeset="/d/display/my_trigger" required="true()"/>
```

Bind Attributes

The following attributes are supported on `<bind>` nodes. Only the nodeset attribute is required.

attribute	description
`nodeset`	Specifies the path to the instance node' [required].
`type`	Specifies the data type. These are discussed below. Considered string if omitted.
`readonly`	Specifies whether the user is allowed to enter data, options: `true()`, and `false()`. Considered false() if omitted.
`required`	Specifies whether the question requires a non-empty value, options: `true()`, and `false()`. Considered false() if omitted.
`relevant`	Specifies whether the question or group is relevant. The question or group will only be presented to the user when the XPath expression evaluates to true. When false the data node (and their descendants) is/are emptied.
`constraint`	Specifies acceptable answers for the specified prompt with an XPath expression.
`calculate`	Calculates a node value with an XPath expression.
`saveIncomplete`	Specifies whether to automatically save the draft record when the user reaches this question, options `true()` and `false()`. Considered false() if omitted.
`jr:constraintMsg`	The message that will be displayed if the specified constraint is violated.
`jr:preload`	Preloaders for predefined meta data. See preloaders.
`jr:preloadParams`	Parameters used by `jr:preload`. See preloaders.

Data Types

type	description
`string`	As in XML 1.0
`int`	As in XML 1.0
`boolean`	As in XML 1.0

`decimal`	As in XML 1.0
`date`	As in XML 1.0
`time`	As in XML 1.0
`dateTime`	As in XML 1.0 review
`select`	space-separated list of strings review
`select1`	as string (spaces strongly discouraged) review
`geopoint`	space-separated list of valid latitude (decimal degrees), longitude (decimal degrees), altitude (decimal meters) and accuracy (decimal meters)
`geotrace`	semi-colon separated list of at least 2 geopoints, where the last geopoint's latitude and longitude is not equal to the first
`geoshape`	semi-colon separated list of at least 3 geopoints, where the last geopoint's latitude and longitude is equal to the first
`binary`	review
`barcode`	as string review

XPath Paths

XPath paths are used in XForms to reference instance nodes to store or retrieve data. Both absolute and relative paths are supported, along with using the proper relative path context node, depending on the situation. Paths can currently only reference XML elements (not attributes, comments, or raw text). The references `.` and `..` are also supported at any point in the path.

The following are examples of valid paths:

- `.`
- `..`
- `/`
- `node`
- `/absolute/path/to/node`
- `../relative/path/to/node`
- `./relative/path/to/node`
- `another/relative/path`
- `//node`

XPath Expressions

All XPath 1.0 expressions are supported, i.e. `|`, `or`, `and`, `=`, `!=`, `<=`, `<`,

`>=`, `>`. Note that predicate support is very limited (see next section).

XPath Predicates

Only the `path/to[node=value]` predicate is supported.

[What about `xpath/to/node[2]`, `xpath/to/node[@attr=value]`, `xpath/to/node[position()=2]` in JavaRosa?. review]

XPath Axes

Only the *parent* and *child* axes are supported of the XPath 1.0 axes. review

XPath Functions

A subset of XPath 1.0 functions, some functions of later versions of XPath, and a number of additional custom functions are supported. Some of the XPath 1.0 functions have been extended with additional functionality.

function	description
`concat(* arg*)`	Deviates from XPath 1.0 in that it may contain *1 argument* and that all arguments can be *nodesets* or strings. It concatenates all string values and *all node values* inside the provided nodesets.
`selected(string list, string value)`	Checks if value is equal to an item in a space-separated list (e.g. `select` data type values).
`selected-at(string list, int index)`	Returns the value of the item at the 1-based index of a space-separated list or empty string if the item does not exist (including for negative index and index 0).
`count-selected(string list)`	Returns the number of items in a space-separated list (e.g. `select` data type values).
`jr:choice-name(string value, node node)`	Returns the label value in the active language corresponding to the choice option with the given value of a select or select1 question question for the given data node. (sorry)
`indexed-repeat(nodeset arg, nodeset repeat1, int index1, [nodeset repeatN, int`	Returns a single node from a nodeset by selecting the 0-based index of a repeat nodeset. It does this up to 3 repeat levels deep

`indexN]{0,2})`	
`true()`	As in XPath 1.0.
`false()`	As in XPath 1.0.
`boolean(* arg)`	As in XPath 1.0.
`boolean-from-string(string arg)`	Returns true if arg is “true” or “1”, otherwise returns false.
`not(boolean arg)`	As in XPath 1.0.
`number(* arg)`	As in XPath 1.0.
`decimal-date(date value)`	Converts date value to a number.
`decimal-date-time(dateTime value)`	Converts dateTime value to a number.
`decimal-time(time value)`	Converts time value to a number.
`int(* arg)`	Converts to an integer.
`string(* arg)`	As in XPath 1.0.
`format-date(date value, string format)`	Returns the date value formatted as defined by the format argument using the following identifiers: `%Y` : 4-digit year `%y` : 2-digit year `%m` 0-padded month `%n` numeric month `%b` short text month (Jan, Feb, etc) `%d` 0-padded day of month `%e` day of month `%H` 0-padded hour (24-hr time) `%h` hour (24-hr time) `%M` 0-padded minute `%S` 0-padded second `%3` 0-padded millisecond ticks `%a` short text day (Sun, Mon, etc)
`date (* value)`	Converts to date.
`regex(string`	Returns result of regex test on provided value. The regular

`value, string expression)`	expression is created from the provided expression string (`'[0-9]+'` becomes `/[0-9]+/`).
`coalesce(string arg1, string arg2)`	Returns first non-empty value of arg1 and arg2 or empty if both are empty and/or non-existent.
`join(string separator, nodeset nodes*)`	Joins the provided arguments using the provide separator between values.
`substr(string value, number start, number end?)`	Returns the substring beginning at the specified *0-based* start index and extends to the character at end index - 1.
`string-length(string arg)`	Deviates from XPath 1.0 in that the argument is *required*.
`count(nodeset arg)`	As in XPath 1.0.
`sum(nodeset arg)`	As in XPath 1.0.
`max(nodeset arg*)`	As in XPath 2.0. pending
`min(nodeset arg*)`	As in XPath 2.0. pending
`round(number arg, number decimals?)`	Deviates from XPath 1.0 in that a second argument may be provided to specify the number of decimals. pending
`pow(number value, number power)`	As in XPath 3.0.
`today()`	Returns today's datetime as a string review
`now()`	same as today() review
`random()`	Returns a random number between 0.0 (inclusive) and 1.0 (exclusive)
`uuid()`	Return a random RFC 4122 version 4 compliant UUID string review
`checklist(number`	Check wether the count of answers that evaluate to true (when it

`min, number max, string v*)`	converts to a number > 0) is between the minimum and maximur inclusive. Min and max can be -1 to indicate *not applicable*.
`weighted-checklist(number min, number max, [string v, string w]*)`	Like checklist(), but the number of arguments has to be even. Each v argument is paired with a w argument that *weights* each v (true) count. The min and max refer to the weighted totals.
`position(node arg?)`	Deviates from XPath 1.0 in that it accepts an argument. This argument has to be a single node. If an argument is provided the function returns the position of that node amongst its siblings (wi the same node name). review
`property(string prop)`	Tbd, this is **not** a valid XPath function in its current JavaRosa implementation. pending
`instance(string id)`	Returns a secondary instance node with the provided id, e.g. `instance('cities')/item/[country=/data/country]` It is the only way to refer to a node outside of the primary instance. Note that it doesn't switch the XML Document (the primary instance) or document root for other expressions. E.g. `/data/country` still refers to the primary instance.
`current()`	In the same league as `instance(ID)` but always referring to the primary instance (and accepting no arguments). Unlike instance(ID), which always requires an absolute path, current() can be used with relative references (e.g. `current()/.` and `current()/..`).
`area(node-set ns | geoshape gs)`	Returns the calculated area in m2 of either a nodeset of geopoin or a geoshape value (not a combination of both) on Earth. It take into account the circumference of the Earth around the Equator but does not take altitude into account.
`once(* calc)`	The parameter will be returned if the context nodes's value is empty, otherwise the current value of the context node will be returned. The function is used e.g. to ensure that a random number is only generated once with `once(random())`. review

Preloaders - Metadata

To be discussed. Not complete and not correct.

review

jr:preload	jr:preloadParams	node	description

instance		/meta/instanceID	Unique Instance ID generated by concatenating ‘uuid:’ with the value generated by the uuid() XPath function.
timestamp	start	/meta/timeStart	Timestamp in datetime data format when the user opened the form. Only populated once.
timestamp	end	/meta/timeEnd	Timestamp in datetime data format when the user last saved the form.
property	deviceid	tbc	Unique identifier of device. Guaranteed not to be blank but could be ‘not supported’. Either the cellular IMEI (with imei: prefix, e.g. imei:A0006F5E212), WiFi mac address (with mac: prefix, e.g mac:01:23:45:67:89:ab), Android ID (e.g. android_id:12011110), or another unique device ID for a webbased client (with domain prefix,e .g. enketo.org:SOMEID).
property	email	tbc	Populate with the user’s email address if the client has access to this, otherwise ‘not supported’.
property	username	tbc	Populate with username stored in the client. Can be blank if not set or ‘not supported’.
property	phone number	tbc	Populate with phone number of device. Can be blank if not available (e.g. for desktop or tablet) or ‘not supported’.
property	simserial	tbc	SIM serial number of phone. May be blank (e.g. for desktop or tablet or ‘not supported’).

property	subscriberid	tbc	IMSI of phone prefixed (with imsi: prefix, e.g. imsi:SD655E212). May be blank (e.g., tablets).

Body

The `<body>` contains the information required to display a question to a user, including the type of prompt, the appearance of the prompt (widget), the labels, the hints and the choice options.

```
<h:body>
    <input ref="/data/firstname">
        <label>What is your first name?</label>
    </input>
    <input ref="/data/lastname">
        <label>What is your last name?</label>
    </input>
    <input ref="/data/age">
        <label>What is your age?</label>
    </input>
</h:body>
```

Body Elements

The following form control elements are supported:

control	description
`<input>`	Used to obtain user input for data types: string, integer, decimal, and date.
`<select1>`	Used to display a single-select list (data type: select1)
`<select>`	Used to display a multiple-select list (data type: select)
`<upload>`	Used for image, audio, and video capture
`<trigger>`	Used to obtain user confirmation (e.g. by displaying a single tickbox or button). Will add value *"OK"* to corresponding instance node when user confirms. If not confirmed the value remains empty. review

Within the user controls the following elements can be used:

element	description

`<group>`	Child of `<body>`, another `<group>`, or a `<repeat>` that groups form controls together. See groups section for further details.
`<repeat>`	Child of `<body>` or `<group>` that can be repeated. See repeats for further details.
`<label>`	Child of a form control element, `<item>`, `<itemset>` or `<group>` used to display a label. Only 1 `<label>` per form control is properly supported but can be used in multiple languages).
`<hint>`	Child of a form control element used to display a hint. Only 1 `<hint>` element per form control is properly supported but can be used in multiple languages).
`<output>`	Child of a `<label>` or `<hint>` element used to display an instance value.
`<item>`	Child of `<select>` or `<select1>` that defines an choice option.
`<itemset>`	Child of `<select>` or `<select1>` that defines a list of choice options to be obtained elsewhere (from a secondary instance).
`<value>`	Child of `<item>` or `<itemset>` that defines a choice value.

Body Attributes

The following attributes are supported on body elements. Note that most attributes can only be used on specific elements. If such a specific attribute is used on elements that do not support it, it will usually be silently ignored.

attribute	description
`ref` / `nodeset`	To link a body element with its corresponding data node and binding, both `nodeset` and `ref` attributes can be used. The convention that is helpful is the one used in XLSForms: use `nodeset="/some/path"` for `<repeat>` and `<itemset>` elements and use `ref="/some/path"` for everything else. The `ref` attribute can also refer to an itext reference (see languages)
`class`	Equivalent to class in HTML and allows a list of space-separate css classes as value. This attribute is only supported on the `<h:body>` element for form-wide style classes.

`appearance`	For all form control elements and groups to change their appearance. See appearances
`jr:count`	For the `<repeat>` element (see repeats). This is one of the ways to specify how many repeats should be created by default.
`jr:noAddRemove`	For the `<repeat>` element (see repeats). This indicates whether the user is allowed to add or remove repeats. Can have values `true()` and `false()`
`autoplay`	For all 5 form control elements, this automatically plays a video or audio 'label' if the question is displayed on its own page, when the user reaches this page.
`accuracyThreshold`	For `<input>` with type `geopoint`, `geotrace`, or `geoshape` this sets the auto-accept threshold in meters for geopoint captures. review
`rows`	Specifies the minimum number of rows a string `<input>` field gets in ODK Collect. In Enketo a similar effect is achieved by adding appearance="multiline". pending

Appearances

The appearance of the 5 form controls can be changed with the appearance attributes. Appearance values usually relate to a specific data type. See the XLS Form specification for a list of appearance attributes are available for each data type. Multiple space-separated appearance values can be added to a form control

TO ADD: 3rd party app launching with an appearance review

Groups

A `<group>` combines elements together. If it has a child `<label>` element, the group is considered a *presentation group* and will be displayed as a visually distinct group.

A `<group>` may or may not contain a `ref` attribute. If it does, the group is considered a *logical group*. A logical group has a corresponding element in the primary instance and usually a corresponding `<bind>` element. A logical group's `ref` is used as the context node for the relative `ref` paths of its descendants.

A group can be both a logical and a presentation group.

Groups may be nested to provide different levels of structure.

Apart from providing structure, a logical group can also contain a `relevant` attribute on its `<bind>` element, offering a powerful way to keep form logic maintainable (see bind attributes).

The sample below includes both the body and corresponding instance. The respondent group is a logical group and the context group is both a logical and a presentation group. The context group will only be shown if both first name and last name are filled in.

```
<h:head>
    <h:title>My Survey</h:title>
    <model>
        <instance>
            <data id="mysurvey">
                <respondent>
                    <firstname/>
                    <lastname/>
                    <age/>
                </respondent>
                <context>
                    <location/>
                    <township/>
                    <population/>
                </context>
                <meta>
                    <instanceID/>
                </meta>
            </data>
        </instance>
        ....
        <bind nodeset="/data/context"
              relevant="string-length(../respondent/firstname)
               string-length(../respondent/lastname) > 0" />
        ....
    </model>
</h:head>
<h:body>
    <group ref="/data/respondent">
        <input ref="firstname">
          <label>What is your first name?</label>
        </input>
        <input ref="lastname">
          <label>What is your last name?</label>
        </input>
        <input ref="age">
          <label>What is your age?</label>
        </input>
    </group>
    <group ref="/data/context">
```

```
        <label>Context</label>
        <input ref="location">
          <label>Record the location</label>
        </input>
        <input ref="township">
          <label>What is the name of the township</label>
        </input>
        <input ref="population">
          <label>What is the estimated population size</label>
        </input>
    </group>
</h:body>
```

Repeats

Repeats are sections that may be repeated in a form. They could consist of a single question or multiple questions. It is recommended to wrap a `<repeat>` inside a `<group>` though strictly speaking not required.

A `<repeat>` uses the nodeset attribute to identify which instance node (and its children) can be repeated.

```
...
<h:head>
    <h:title>A Survey with repeats</h:title>
    <model>
        <instance>
            <data id="repeats" version="2014083101">
                <person>
                    <name />
                    <relationship />
                </person>
                <meta>
                    <instanceID/>
                </meta>
            </data>
        </instance>
        ...
    </model>
</h:head>
<h:body>
    <group ref="/data/person">
        <label>Person</label>
        <repeat nodeset="/data/person">
            <input ref="/data/person/name">
                <label>Enter name</label>
            </input>
```

```
            <input ref="/data/person/relationship">
                <label>Enter relationship</label>
            </input>
        </repeat>
    </group>
</h:body>
...
```

Creation, Removal of Repeats

The default behaviour of repeats is to let the user create or remove repeats using the the user interface. ODK Collect will ask for the first repeat. Enketo will show the first repeat automatically. This can be disabled by adding the attribute `jr:noAddRemove="true()"` to the `<repeat>` element.

There are 2 different ways to ensure that multiple repeats are automatically created when a form loads.

A. Multiple nodes can be defined in the primary instance of the XForm. E.g. see below for an instance that will automatically create 3 repeats for the above form.

```
...
<instance>
    <data id="repeats" version="2014083101">
        <person>
            <name />
            <relationship />
        </person>
        <person>
            <name />
            <relationship />
        </person>
        <person>
            <name />
            <relationship />
        </person>
        <meta>
            <instanceID/>
        </meta>
    </data>
</instance>
...
```

B. Using the `jr:count` attribute on the `<repeat>` element. E.g. see below for the use of jr:count to automatically create 3 repeats for the above form. The value could also be a `/path/to/node` and clients should evaluate the number of repeats dynamically (Note: It is problematic to implement this in a truly dynamic fashion, i.e. when the value changes, to update the number of repeats).

```
...
<h:body>
    <group ref="/data/person">
        <label>Person</label>
        <repeat nodeset="/data/person" jr:count="3">
            <input ref="/data/person/name">
                <label>Enter name</label>
            </input>
            <input ref="/data/person/relationship">
                <label>Enter relationship</label>
            </input>
        </repeat>
    </group>
</h:body>
...
```

Default Values in Repeats

There are two different ways to provide default values to elements inside repeats.

A. Specify the values inside a repeat group with a `jr:template=""` attribute in the primary instance. Any new repeat that does not yet exist in the primary instance will get these default values. The repeat group with the `jr:template` attribute is **not** part of the record itself. So in the example below is for a form in which only a single repeat was created for John.

```
...
<instance>
    <data id="repeats" version="2014083101">
        <person jr:template="" >
            <name />
            <relationship>spouse</relationship>
        </person>
         <person>
            <name>John</name>
            <relationship>father</relationship>
        </person>
        <meta>
            <instanceID/>
        </meta>
    </data>
</instance>
...
```

B. Specify the values for each repeat instance individually in the primary instance. In the example below the form will be loaded with 2 repeats with the values for John and Kofi.

```
...
<instance>
    <data id="repeats" version="2014083101">
        <person>
            <name>John</name>
            <relationship>father</relationship>
        </person>
        <person>
            <name>Kofi</name>
            <relationship>brother</relationship>
        </person>
        <meta>
            <instanceID/>
        </meta>
    </data>
</instance>
...
```

A Big Deviation with XForms

In XForms, relative XPaths should be evaluated *relative to context*, and absolute paths (/data/path/to/repeat) should be evaluated as *absolute paths without considering context*. If there are multiple repeats, the XPath /data/path/to/repeat would either return the first repeat (if e.g. a string value is requested), or all repeats (if a nodeset is requested).

However, in this spec, due to an unfortunate persistent historical error, **absolute paths** /data/path/to/repeat/node **inside repeats are always evaluated as if they are relative to the current nodeset**. In other words, the absolute XPath `/data/path/to/repeat/node` when it is referred to from inside a repeat is evaluated as if it is the relative XPath `../node`.

In order to rectify this error at some time in the future, it would be very helpful if any form builders around this spec start generating relative references automatically.

Languages

Multi-lingual content for labels, and hints is supported. This is optional and can be done by replacing all language-dependent strings with ‘text identifiers’, which act as indexes into a multi-lingual dictionary in the model.

In the `<model>`, a multi-lingual dictionary has the following structure:

```
<itext>
    <translation lang="[language name]" default="true()">
        <text id="[text id]">
            <value>[translation of text with [text id]]</value>
        </text>
```

```
    </translation>
</itext>
```

Additional `<text>` entries are added for each localizable string. The `<translation>` block is duplicated for each supported language. The content should be the same (same set of text ids) but with all strings translated to the new language. The language name in the lang attribute should be human-readable, as it is used to identify the language in the UI. A default="" attribute can be added to a `<translation>` to make it the default language, otherwise the first listed is used as the default. Every place localized content is used (all `<label>`s and `<hint>`s) must use a converted notation to reference the dictionary:

For example:

```
<label>How old are you?</label>
```

is changed to:

```
<label ref="jr:itext('how-old')" />
```

With the corresponding entries in `<itext>`:

```
<translation lang="English">
    ...
    <text id="how-old">
        <value>How old are you?</value>
    </text>
    ...
</translation>
<translation lang="Spanish">
    ...
    <text id="how-old">
        <value>¿Cuantos años tienes?</value>
    </text>
    ...
</translation>
...
```

Not every string must be localized. It is acceptable to intermix `<label>`s of both forms. Those which do not reference the dictionary will always show the same content, regardless of language.

In general, all text ids must be replicated across all languages. It is sometimes only a parser warning if you do not, but it will likely lead to headaches. Even within a single language, it is helpful to have multiple 'forms' of the same string. For example, a verbose phrasing used as the caption when answering a question, but a short, terse phrasing when that question is shown in the form summary. We handle this as follows:

review

```
<text id="how-old">
    <value form="long">How old are you?</value>
    <value form="short">Age</value>
</text>
```

The different `forms` are only supported for question captions (`<label>`s inside user controls). The media section describes how to add non-text form labels using the same method.

Media

The `<itext>` method described in the languages section can also be used for **media labels**. Media labels can be used in addition to text labels or instead of text labels.

```
....
<itext>
    <translation default=true() lang="English">
        <text id="/widgets/select_widgets/grid_test/b:label">
            <value form="image">jr://images/b.jpg</value>
        </text>
        <text id="/widgets/display_widgets/text_media:label">
            <value form="audio">jr://audio/goldeneagle.mp3</val
            <value>You can add a sound recording.</value>
        </text>
    </translation>
</itext>
...
```

Supported Media Types

- “image”
- “audio”
- “video”
- “big-image”

By default, itext “image” values are not clickable. However, if you also include a “big-image”, the image displayed by “image” will be clickable and will display a pannable, zoomable view of the file specified by “big-image”. The user interface must provide a way to go back to the form after opening a “big-image”. Specifying “big-image” alone has no effect, you must always include “image”.

Files referenced by “image” and “big-image” may be the same; however, for performance reasons, it is recommended to create smaller thumbnail images to be referenced by “image”.

Future

The following desired future features have been identified.

- support for external data to create itemsets (like a secondary instance but loaded from an external source)
- a way to specify a slider widget with min, max and step for numerical inputs
- a better way to record and retrieve metadata

Apps

These are two examples of data collection applications and libraries that are conforming to this specification:

- Enketo Core
- ODK Collect

Acknowledgements

The JavaRosa XForms document maintained by Dimagi formed the basis of this document.