Introduction

Changelog

The ODK XForms specification is used by tools in the ODK ecosystem. It is a subset of the far larger W3C XForms 1.0 specification and also contains a few additional features not found in the W3C XForms specification.

The purpose of this specification is to provide a common form description standard that many different kinds of compatible tools can be based on. Using a single, shared form description standard has the following advantages:

1. Users in the ODK ecosystem can mix and match tools and reassess which they use based on their changing needs. In particular, they don't get locked in to tools that may become deprecated or for which an attractive replacement becomes available.
2. Tool implementors in the ODK ecosystem can benefit from feedback from a broad range of collaborators when designing new core functionality.
3. Tool implementors in the ODK ecosystem can share core implementations.

This document is intended primarily for developers who build form processing engines or software form builders. Most organizations who use tools in the ODK ecosystem for data collection will prefer to create forms using the XLSForm standard or a graphical form builder.

A version of this specification was initially developed by the OpenRosa Consortium. JavaRosa is a Java library initially developed by the consortium as a J2ME app that implements this specification. There are now several other compatible implementations.

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 XForm:

```
<?xml version="1.0"?>
<h:html xmlns="http://www.w3.org/2002/xforms"
        xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:jr="http://openrosa.org/javarosa"
        xmlns:orx="http://openrosa.org/xforms"
        xmlns:odk="http://www.opendatakit.org/xforms"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <h:head>
        <h:title>My Survey</h:title>
        <model>
            <instance>
                <data id="mysurvey" orx:version="2014083101">
                    <firstname></firstname>
                    <lastname></lastname>
                    <age></age>
                    <orx:meta>
                        <orx:instanceID/>
                    </orx:meta>
                </data>
            </instance>
            <bind nodeset="/data/firstname" type="xsd:string" required="true()" />
            <bind nodeset="/data/lastname"  type="xsd:string" />
            <bind nodeset="/data/age" type="xsd:int" />
            <bind nodeset="/data/orx:meta/orx:instanceID" preload="uid" type="xsd:string"/>
        </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>
```

Outside of this simplified structure there are ways to define:

- form title as the `<title>` element, a child of the `<head>` element in the same "http://www.w3.org/1999/xhtml" namespace,
- linkages with external (mobile) applications,
- language dictionaries.

Namespaces

XML namespaces provide a way to avoid name conflicts for element and attribute names. In ODK XForms, the elements and attributes that are also in XForms 1.0 are in the XForms namespace which is declared as the default namespace in the example above (`xmlns="http://www.w3.org/2002/xforms"`). Setting a default namespace means that non-prefixed elements and attributes are assigned that namespace.

Elements and attributes that are specific to ODK XForms and not defined by the XForms 1.0 specification should be separately namespaced. For historical reasons, the `"http://openrosa.org/javarosa"` namespace (with the `jr` prefix in this document), and the `"http://openrosa.org/xforms"` namespace (with the `orx` prefix in this document) have been used.

For any new additions not defined in another specification, the `"http://www.opendatakit.org/xforms"` namespace is now preferred. It is assigned the `odk` prefix in this documentation. If a new feature is copied from another XForms implementation the originator's namespace will be used.

For more information about namespaces, see the XML Namespaces specification.

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* is the first instance defined by the form and 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" orx: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 rendered. For nodes of type "binary", defaults use file endpoint URIs.

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 by the server that publishes the Form and receives data submissions. For more information see the OpenRosa Form List API. [required]
`orx:version`	on the childnode of the primary instance in the *http://openrosa.org/xforms/* namespace: Form version which can contain any string value. Like meta nodes this information is used as a *processing cue* for the server receiving the submission.
`odk:prefix`	on the childnode of the primary instance in the *http://opendatakit.org/xforms* namespace: optional string prefix which is included at the beginning of the compact representation
`odk:delimiter`	on the childnode of the primary instance in the *http://opendatakit.org/xforms* namespace: optional string delimiter which is used to separate prefix, tags and values in the compact representation
`odk:tag`	on a question node (grandchild of the primary instance) in the *http://opendatakit.org/xforms* namespace: optional string tag which is used to identify nodes that should be part of the compact representation
`jr:template`	on any repeat group node in the *http://openrosa.org/javarosa namespace*: 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 and only affects the behavior of the form engine. For more details, see the repeats section.

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

Secondary Instances - Internal

Secondary instances are used to pre-load read-only data inside a form. This data is searchable in XPath. At the moment the key use case is in designing 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 `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>
```

Secondary Instances - External

The previous section discussed secondary instances with static read-only data that is present in the XForm document itself. Another type of secondary instances presents read-only data from an *external* source. The external source can be static or dynamic and is specified using the additional `src` attribute with a URI value on an empty `<instance>` node. Querying an external instance is done in exactly the same way as for an internal secondary instance.

```
<instance id="countries" src="jr://file/country-data.xml"/>
```

See the section on URIs for acceptable URI formats that refer to an external secondary 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.

An instance node does not require a corresponding `<bind>` node, regardless of whether it has a presentation node.

```
<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=". &lt; 10" jr:constraintMsg
<bind nodeset="/d/number_widgets/my_decimal" type="decimal" constraint=". &gt; 10.51 and . 
<bind nodeset="/d/dt/my_date" type="date" constraint=". &gt;= today()" jr:constraintMsg="on
<bind nodeset="/d/dt/my_time" type="time"/>
<bind nodeset="/d/dt/dateTime" type="dateTime"/>
<bind nodeset="/d/s/my_select" type="select" constraint="selected(., 'c') and selected(., '
<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`	As in XForms 1.0 this specifies the path to the instance node or attribute [required].
`type`	As in XForms 1.0 this specifies the data type. These data types values are supported and is considered "string" if omitted or if an unknown type is provided.
`readonly`	As in XForms 1.0 this specifies whether the user is allowed to enter data, using a boolean expression. Considered `false()` if omitted.
`required`	As in XForms 1.0 this pecifies whether the question requires a non-empty value, using a boolean expression. Considered `false()` if omitted.
`relevant`	As in XForms 1.0 this 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 its descendants) are removed from the primary instance on submission.
`constraint`	As in XForms 1.0 this specifies acceptable answers for the specified prompt with an XPath expression. Will only be evaluated when the node is non-empty.
`calculate`	As in Xforms 1.0 this 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:requiredMsg`	Specifies the custom message to be displayed when the `required` is violated. Value can be string literal (`jr:constraintMsg="message"`) or a translation function call (`jr:constraintMsg="jr:itext('id')"`).
`jr:constraintMsg`	Specifies the custom message to be displayed when the `constraint` is violated. Value can be string literal (`jr:requiredMsg="message"`) or a translation function call (`jr:requiredMsg="jr:itext('id')"`).
`jr:preload`	Preloaders for predefined meta data. See preload attributes.
`jr:preloadParams`	Parameters used by `jr:preload`. See preload attributes.
`orx:max-pixels`	Specifies a transformation for uploaded images (binary datatype), e.g. `orx:max-pixels="1024"`. If the long edge of the image is larger than the provided number value, the image should be resized proportionally so that the long edge matches the provided pixel value.

Data Types

The following are acceptable data type values.

type	description
`string`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`int`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`boolean`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`decimal`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`date`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`time`	As in XML 1.0, optionally in "http://www.w3.org/2001/XMLSchema" namespace
`dateTime`	Deviates from XML 1.0, in that it *includes the timezone offset* (i.e. not normalized to UTC). The timezone offset is HH:MM, where both hours and minutes are required and are zero-padded, preceded by the + or - sign without any spaces. The offset may also equal "Z".
`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`	URI pointing to binary file. For user-uploaded files attached to a submission, only the filename with extension should be used without a scheme or subdirectories.
`barcode`	As string
`intent`	As string, used for external applications

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 Operators

All XPath 1.0 operators are supported, i.e. `|`, `and`, `or`, `mod`, `div`, `=`, `!=`, `<=`, `<`, `>=`, `>`, `+`, `-`.

Note that the standard XPath type conversions are extended by this specification in the `number()` function. This extended functionality provides the ability to perform arithmetic with, and compare, date and dateTime strings.

XPath Predicates

Predicates are fully supported but with the limitations described in XPath Axes and XPath Functions

XPath Axes

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

XPath Functions

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

The XPath evaluator will automatically cast function arguments to their required data types by calling the `number()`, `string()`, `boolean()` functions, as described in XPath 1.0. The XPath evaluator has no knowledge of the data type of the value stored in the model. In XForms, node values are always stored and obtained as strings.

Note: since expression results are stored in the XForms model as strings using the `string()` function, a boolean `false` result, such as from the expression `1 > 2`, is stored in the model as the string `"false"`. When referring to that node in another expression as a boolean argument, the ***string value*** *of that node ("false") is converted to a boolean by calling the `boolean()` function which returns the boolean `true` because `boolean("false") = true()`. To deal with this, it usually best to not do boolean comparisons with stored values (compare strings instead) or use `boolean-from-string()` in the XPath comparison expression.*

The table below describes the functions, and the data types of their arguments and return values, using the following special argument characters:

- `?` argument is optional
- `*` argument can be repeated
- `|` alternative argument is allowed

For convenience, the functions are categorized based on their main usage. Some functions could be argued to (also) belong in another category. However, the data type rules mentioned above are the same for all functions, regardless of the category they have been placed under.

function	returns	description

String Functions		
`string(* arg)`	string	As in XPath 1.0.
`concat(string arg*|node-set arg*)`	string	Deviates from XPath 1.0 in that it may contain *1 argument* and that all argument can be *node-sets* or strings. It concatenates all string values and *all node values* inside the provided node-sets.
`join(string separator, node-set nodes*)`	string	Joins the provided arguments using the provide separator between values.
`substr(string value, number start, number end?)`	string	Returns the substring beginning at the specified *0-based* start index and extends to the character at end index - 1.
`substring-before(string, string)`	string	As in XPath 1.0.
`substring-after(string, string)`	string	As in XPath 1.0.
`translate(string, string, string)`	string	As in XPath 1.0.
`string-length(string arg)`	number	Deviates from XPath 1.0 in that the argument is *required*.
`normalize-space(string arg?)`	string	As in XPath 1.0
`contains(string haystack, string needle)`	boolean	As in XPath 1.0.
`starts-with(string haystack, string needle)`	boolean	As in XPath 1.0.
`ends-with(string haystack, string needle)`	boolean	As in XPath 3.0.
`uuid(number?)`	string	Without arguments, it returns a random RFC 4122 version 4 compliant UUID. Wit an argument it returns a random string with the provided number of characters.
`digest(string src, string algorithm, string encoding?)`	string	As in XForms 1.1
`pulldata(string instance_id, string desired_element, string query_element, string query)`	string	Returns a single value from a secondary instance based on the specified query. Shortcut for `instance(instance_id)/root/item[query_element=query]/desired_elemen`
Boolean Functions		
`if(boolean condition, * then, * else)`	string	Deviates from XForms 1.0 in that the 2nd and 3rd parameter are objects and not strings.
`coalesce(string arg1, string arg2)`	string	Returns first non-empty value of arg1 and arg2 or empty if both are empty and/o non-existent.
`once(string`	string	The parameter will be evaluated and returned if the context nodes’s value is

`calc)`		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 wit `once(random())`.
`true()`	boolean	As in XPath 1.0.
`false()`	boolean	As in XPath 1.0.
`boolean(* arg)`	boolean	As in XPath 1.0.
`boolean-from-string(string arg)`	boolean	Deviates from XForms 1.0 in that it returns `false` for any argument that is not “true” or “1”.
`not(boolean arg)`	boolean	As in XPath 1.0.
`regex(string value, string expression)`	boolean	Returns result of regex test on provided value. The regular expression is created from the provided expression string (`'[0-9]+'` becomes `/[0-9]+/`).
`checklist(number min, number max, string v*)`	boolean	Check whether the count of answers that evaluate to true (when it converts to a number > 0) is between the minimum and maximum inclusive. Min and max can be -1 to indicate *not applicable*.
`weighted-checklist(number min, number max, [string v, string w]*)`	boolean	Like checklist(), but the number of arguments has to be even. Each v argument i paired with a w argument that *weights* each v (true) count. The min and max ref to the weighted totals.
Number Functions		
`number(* arg)`	number	As in XPath 1.0. In addition it will convert date- and dateTime-formatted strings t a number of days since January 1, 1970 UTC.
`random()`	number	Deviates from XForms 1.1 by not supporting a parameter.
`int(number arg)`	number	Converts to an integer (a whole number) by discarding the fractional component a number.
`sum(node-set arg)`	number	As in XPath 1.0.
`max(node-set arg*)`	number	As in XPath 3.0.
`min(node-set arg*)`	number	As in XPath 3.0.
`round(number arg, number decimals?)`	number	Deviates from XPath 1.0 in that a second argument may be provided to specify t number of decimals.
`pow(number value, number power)`	number	As in XPath 3.0.
`log(number arg)`	number	As in XPath 3.0.
`log10(number arg)`	number	As in XPath 3.0.
`abs(number arg)`	number	As in XPath 3.0.
`sin(number arg)`	number	As in XPath 3.0.
`cos(number arg)`	number	As in XPath 3.0.
`tan(number arg)`	number	As in XPath 3.0.
`asin(number arg)`	number	As in XPath 3.0.
`acos(number arg)`	number	As in XPath 3.0.
`atan(number arg)`	number	As in XPath 3.0.
`atan2(number arg, number arg)`	number	As in XPath 3.0.
`sqrt(number arg)`	number	As in XPath 3.0.

`exp(number arg)`	number	As in XPath 3.0.
`exp10(number arg)`	number	As in XPath 3.0.
`pi()`	number	As in XPath 3.0.
Node-set Functions		
`count(node-set arg)`	number	As in XPath 1.0.
`count-non-empty(node-set arg)`	number	As in XForms 1.0.
`position(node arg?)`	number	Deviates from XPath 1.0 in that it accepts an argument. This argument has to be single node. If an argument is provided the function returns the position of that node amongst its siblings (with the same node name).
`instance(string id)`	node-set	As in XForms 1.0. Note that it doesn't switch the document root for predicates. E in `instance('cities')/item/[country=/data/country]`, the `/data/countr` path still refers to the primary instance.
`current()`	node-set	As in XForms 1.1. Used inside predicates of expressions that use instance() to enable referring to a node relative to the context of the *current* question. E.g. as `instance('countries')/item[name=current()/../name]/capital`).
`randomize(node-set arg, number seed)`	node-set	Shuffles the node-set argument using the "inside-out" variant of the Fisher-Yates algorithm. The optional seed argument performs a (reproducible) shuffle using th same algorithm with a *seeded* Park Miller Pseudo Number Generator.
Date and Time Functions		
`today()`	string	Returns a string with today's local date in the format described under the date datatype.
`now()`	string	Deviates from XForms 1.0 in that it returns the current date and time *including timezone offset* (i.e. not normalized to UTC) as described under the dateTime datatype.
`format-date(date value, string format)`	string	Returns the provided 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 `%a` short text day (Sun, Mon, etc).* * If form locale can be determined that locale will be used. If form locale cannot b determined the locale of the client will be used (e.g. the browser or app).
`format-date-time(dateTime value, string format)`	string	Returns the provided dateTime value formatted as defined by the format argume using the same identifiers as `format-date` plus the following: `%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.* * If form locale can be determined that locale will be used. If form locale cannot b determined the locale of the client will be used (e.g. the browser or app).
`date(* value)`	string	Converts to a string in the ....date format.
`decimal-date-time(dateTime value)`	number	Converts dateTime value to the number of days since January 1, 1970 UTC.
`decimal-time(time value)`	number	Converts time value to a number representing a fractional day in the device's timezone. For example, noon is 0.5 and 6pm is 0.75.
Select Functions		

`selected(string list, string value)`	boolean	Checks if value is equal to an item in a space-separated list (e.g. `select` data type values).
`selected-at(string list, number index)`	string	Returns the value of the item at the 0-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(node node)`	number	Returns the number of items in a space-separated list (e.g. `select` data type values).
`jr:choice-name(node node, string value)`	string	Returns the label value in the active language corresponding to the choice optio with the given value of a select or select1 question for the given data node. (sor
Translation Functions		
`jr:itext(string id)`	string	Obtains an itext value for the provided reference in the active language from the `<itext>` block in the model.
Repeat Functions		
`indexed-repeat(node-set arg, node-set repeat1, number index1, [node-set repeatN, number indexN]{0,2})`	string	Returns a single node value from a node-set by selecting the 1-based index of a repeat node-set that this node is a child of. It does this up to 3 repeat levels dee E.g. `indexed-repeat(//node, /path/to/repeat, //index1, /path/to/repeat/nested-repeat, //index2)` is meant to be a shortcut for `//repeat[position()=//index1]/nested-repeat[position()=index2]/node` in native XPath syntax.
Geographic Functions		
`area(node-set ns|geoshape gs)`	number	Returns the calculated area in m2 of either a node-set of geopoints or a geoshap value (not a combination of both) on Earth. It takes into account the circumferen of the Earth around the Equator but does not take altitude into account.
`distance(node-set ns|geoshape gs|geotrace gt)`	number	Returns the distance in meters of either a node-set of geopoints or a single geoshape value or a single geotrace value (not a combination of these) on Earth, the sequence provided by the points in the parameter. It takes into account the circumference of the Earth around the Equator and does not take altitude into account.

Metadata

This section describes metadata about *the record* that is created with the form. Metadata about *the form itself* (id, version, etc) is covered in the Primary Instance section.

The namespace of the meta block is either the default XForms namespace or “https://openrosa.org/xforms”. The latter is recommended.

```
<instance>
    <data id="myform" orx:version="637">
        <question2/>
        <casename/>
        <confirm/>
        <orx:meta>
            <orx:deviceID/>
            <orx:timeStart/>
            <orx:timeEnd/>
            <orx:userID/>
            <orx:instanceID/>
            <orx:audit/>>
        </orx:meta>
    </data>
</instance>
```

These meta elements have corresponding `<bind>` elements with either a calculation or with *preload attributes*. Note that when using a calculation these values may be recalculated, e.g. when a draft record is loaded. This could lead to undesirable results for example when the result is a random value or timestamp.

Using both a calculation and preload attributes is technically allowed but never recommended, because one will overwrite the other.

The following meta elements are supported:

element	description	default datatype	value	namespace
`instanceID`	The unique ID of the record [required]	string	concatenation of 'uuid:' and uuid()	same as meta block
`timeStart`	A timestamp of when the form entry was started	datetime	now()	same as meta block
`timeEnd`	A timestamp of when the form entry ended	datetime	now()	same as meta block
`userID`	The username stored in the client, when available	string		same as meta block
`deviceID`	Unique identifier of client install. Guaranteed not to be blank. For privacy reasons, this identifier should be stored as application state and be user-resettable (e.g. by reinstalling the client or clearing cookies). Clients typically use a prefix to identify themselves (e.g. `enketo.org:SOMEID`).	string	depends on client, prefixed	same as meta block
`deprecatedID`	The `<instanceID/>` of the submission for which this is a revision. This revision will get a newly generated `<instanceID/>` and this field is populated by the prior value. Server software can use this field to unify multiple revisions to a submission into a consolidated submission record.	string		same as meta block
`email`	The user's email address when available.	string		same as meta block
`phoneNumber`	The phone number of the device, when available	string		same as meta block
`audit`	A CSV or zipped CSV file containing audit logs pertaining to the record (e.g., timing, location). The file is attached in the same way as for an `<upload>` form control and binary instance node. Filename is determined by the client and file follows this documented format. What data is recorded is configurable via audit attributes.	binary	filename	same as meta block

Preload Attributes

As mentioned in Bind Attributes, there are two different preload attributes. A particular combination of preload attributes populates a value according to a **predetermined fixed formula**, when a **predetermined event** occurs. Different combinations handle different events and use a different calculation.

Supported preload attribute combinations are:

jr:preload	jr:preloadParams	value	event
uid		see `instanceID`	odk-instance-first-load
timestamp	start	see `timeEnd`	odk-instance-first-load
timestamp	end	see `timeEnd`	xforms-revalidate
property	deviceid	see `deviceID`	odk-instance-first-load
property	email	see `email`	odk-instance-first-load
property	username	see `userID`	odk-instance-first-load
property	phone number	see `phoneNumber`	odk-instance-first-load

Audit Attributes

attribute	description
`odk:location-priority`	`no-power`, `low-power`, `balanced`, or `high-accuracy` as defined in LocationRequest. Required to enable location in log.
`odk:location-min-interval`	The desired minimum time, in seconds, location updates will be fetched. Required to enable location in log.
`odk:location-max-age`	The maximum time, in seconds, locations will be considered valid. Must be greater than or equal to `odk:location-min-interval`. Required to enable location in log.
`odk:track-changes`	Can be set to `"true"` or `"false"`. If true, whenever an answer is changed, the old value and new value will be added to the log. Attribute is not required and defaults to false.

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>`	This element is used to obtain user input for data types: string, integer, decimal, and date. As in XForms 1.0 without Special Attributes support.
`<select1>`	Used to display a single-select list (data type: string). As in XForms 1.0 without Special Attributes support.
`<select>`	Used to display a multiple-select list (data type: string). As in XForms 1.0 without Special Attributes support.
`<upload>`	Used for image, audio, and video capture. As in XForms 1.0 without support for filename and mediatype child elements, nor the `incremental` attribute and only supporting the `binary` data type.
`<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.
`<range>`	Used to obtain numeric user input from a sequential range of values. Mostly as in XForms 1.0. However, it does not support the `incremental` attribute, and the `step`, `start`, and `end` attributes are required.
`<odk:rank>`	Used to require user to rank/order options. The ordered options are recorded as a space-separated list (as with `<select>`). The recorded list always includes all options.

The following user interface elements are supported:

element	description
`<group>`	Child of `<body>`, another `<group>`, or a `<repeat>` that groups form controls together. See groups section for further details. As in XForms 1.0.
`<repeat>`	Child of `<body>` or `<group>` that can be repeated. See repeats for further details.

Within the form controls the following elements can be used:

element	description

`<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). As in XForms 1.0 without support for Linking Attributes.
`<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). As in XForms 1.0 without support for Linking Attributes.
`<output>`	Child of a `<label>` or `<hint>` element used to display an instance value, inline, as part of the label, or hint text. It can also be a child of a `<text>` translation. As in XForms 1.0 but only supporting the `value` attribute.
`<item>`	Child of `<select>` or `<select1>` or `<odk:rank>` that defines an choice option. As in XForms 1.0.
`<itemset>`	Child of `<select>` or `<select1>` or `<odk:rank>` that defines a list of choice options to be obtained elsewhere (from a secondary instance). As in XForms 1.0.
`<value>`	Child of `<item>` or `<itemset>` that defines a choice value. As in XForms 1.0.

Below is an example of a label, an output, a hint, an itemset and value used together to define a form control:

```
 <group ref="/data/loc">
    <label>Cities</label>
    ...
    <odk:rank ref="/data/loc/cities">
        <label>Rank these cities</label>
        <hint>Rank the cities in <output value="/data/loc/country"/> in order of importance
        <itemset nodeset="randomize(instance('cities')/root/item[country= /data/loc/country
            <value ref="name"/>
            <label ref="label"/>
        </itemset>
    </odk:rank>
</group>
```

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-separated 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 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
`value`	For the `<output>` element to reference the node value to be displayed.
`rows`	Specifies the minimum number of rows a string `<input>` field gets.
`mediatype`	For the `<upload>` element. The string value specifies the kind of media picker that will be displayed. Unlike in XForms 1.0, only one value can be specified.

	Possible values vary by client and examples include `image/*`, `audio/*` and `video/*`. Ignored if `accept` is also specified.
`accept`	For the `<upload>` element. As from the XForms 2.0 wiki: “comma-separated list of suggested media types and file extensions used to determine the possible sources of data to upload.”
`start`	For the `<range>` element. The lower bound of the range. This attribute is required and its value has to be valid for the data type used.
`end`	For the `<range>` element. The upper bound of the range. This attribute is required and its value has to be valid for the data type used.
`step`	For the `<range>` element. The increment between values that can be selected. This attribute is required and its value has to be valid for the data type used.

Appearances

The appearance of all form controls and of a group can be changed with appearance attributes. Appearance values usually relate to a specific data or question type. See the XLS Form specification for a list of appearance attributes that are available for each data type. Multiple space-separated appearance values can be added to a form control in any order.

An appearance value may also work in conjunction with an image label to substantially alter the appearance and behavior of a form control as is e.g. the case with appearance ‘image-map’.

An appearance attribute can also be used to indicate that an external app should be used as a form control.

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) > 0 and
               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.

A `<repeat>` cannot have a label child element. To display a label it should be wrapped inside a `<group>` as shown below:

```
...
<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>
...
```

When a client needs to compactly show a single repeat instance in its user interface (e.g. as a collapsed repeat or a table-of-contents item), it is recommended to show the label of the first child group of that repeat.

Creation, Removal of Repeats

The default behavior of repeats is to let the user create or remove repeats using the user interface. The user control for creating and removing repeats 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.

```
...
<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>
...
```

Events and Actions

XForm Events are dispatched following different steps in the form lifecycle. XForms Actions can be invoked in response to these events. This makes it possible to define exactly when certain tasks should occur.

Events

See the W3C XForms specification section on events. The following events are supported:

event	description
`odk-instance-first-load`	dispatched the first time an instance is loaded
`xforms-value-changed`	As in XForms 1.1.
`odk-new-repeat`	dispatched when a new instance of a repeat is added to the primary instance. See more.

Note: `xforms-ready` *was previously documented as the event dispatched the first time an instance is loaded. Since that definition does not match the W3C XForms event with the same name, it was deprecated in favor of* `odk-instance-first-load`.

Actions

The following subset of actions defined by the W3C XForms specification are supported:

action	description
`setvalue`	Explicitly sets the value of the specified instance data node. See the W3C description. `ref` can be used in place of `bind` to specify a node path instead of a node id.
`odk:setgeopoint`	Sets the current location’s geopoint value in the instance data node specified in the `ref` attribute. Any `value` attribute or textContent will be ignored. Failure to retrieve the location will result in an empty string value.

Action elements triggered by initialization events go in the model as siblings of `bind` nodes. Action elements triggered by control-specific events are nested in that control block. Multiple triggering events may be specified as a space-separated list and in that case, initialization events may be specified in an action element nested in a control block. For example, the value `odk-instance-first-load odk-new-repeat` can be given to the `event` attribute of an action nested in a repeat. That action is then triggered once the first time the primary instance is loaded and every time an instance of the parent repeat is added.

The odk-new-repeat event

The `odk-new-repeat` event is dispatched when a new instance of a repeat is added to the primary instance and before recomputation of `calculates`, `constraints`, etc. Actions triggered by `odk-new-repeat` must be nested in the repeat form control.

The `odk-new-repeat` event is never dispatched for repeat instances that are part of the form definition. However, it is dispatched for repeat instances added by evaluation of the `jr:count` attribute value. See creation, removal of repeats.

The following example demonstrates giving a node in a repeat a default, user-modifiable value based on other user input:

```
<h:head>
    <model>
        ...
        <bind nodeset="/data/my_age" type="int" />
        <bind nodeset="/data/person/age" type="int" />
        <bind nodeset="/data/person/location" />
    </model>
</h:head>
<h:body>
    <input ref="/data/my_age">
        <label>Your age</label>
    </input>
    ...
    <repeat nodeset="/data/person">
        <setvalue event="odk-new-repeat" ref="/data/person/age" value="../../my_age + 2" />
        <odk:setgeopoint event="odk-new-repeat" ref="/data/person/location" />
        <input ref="/data/person/age">
            <label>Person's age</label>
        </input>
        ...
    </repeat>
</h:body>
```

Setting a dynamic value after form load

```
<bind nodeset="/data/now" type="dateTime" />
<bind nodeset="/data/location" />
<setvalue event="odk-instance-first-load" ref="/data/now" value="now()" />
<odk:setgeopoint event="odk-instance-first-load" ref="/data/location" />
```

Setting a static value when a node’s value changes

```
<bind nodeset="/data/my_text" type="string" />
<bind nodeset="/data/my_text_changed" type="string" />
<bind nodeset="/data/my_current_location" type="string" />
...
<input ref="/data/my_text">
    <setvalue event="xforms-value-changed" ref="/data/my_text_changed">Value changed!</setv
    <odk:setgeopoint event="xforms-value-changed" ref="/data/my_current_location" />
</input>
```

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. The language strings can be identified with the `jr:itext()` XPath function.

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.

It is even allowed to intermix both a `ref` and a regular value. In this case, if the itext engine is missing it will refer to the regular value. E.g.

```
<label ref="jr:itext('mykey')">a default value</label>
```

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. This can be done using the `form` attribute, as follows:

```
<text id="how-old-label">
    <value>How old are you?</value>
    <value form="short">Age</value>
</text>
<text id="how-old-hint">
    <value>Enter a number</value>
    <value form="guidance">If the age is less than 18, the remainder of the survey will be
</text>
```

There are three `form` attribute options for text strings:

text type	attribute	description
regular	*none*	Supported for `<label>` and `<hint>` content to display regular labels and hints
short version of label	`short`	Supported for `<label>` content only. It is a shorter version of the label, meant for very small screens, or to be shown in a summary of the form data.
additional guidance hint	`guidance`	Supported for `<hint>` content only. It is a description of the question that can be used to provide further guidance to enumerators. It is not meant to be shown in the client UI by default, but could be shown in a special view mode (e.g., for a training) or on printouts.

The media section describes how to add non-text form labels in a similar manner.

Media

The `<itext>` element 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</value>
            <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”.

URIs

Throughout the XForm format URIs are used to refer to resources outside of the XForm itself. The `jr` scheme is used to indicate the resource is available in a sandboxed environment the client is aware of.

File Endpoints

These URIs point to files. The following are currently supported:

URI format	description
`jr://images/path/to/file.png`	Points to an image resource in the sandboxed environment
`jr://audio/path/to/file.mp3`	Points to an audio resource in the sandboxed environment
`jr://video/path/to/file.mp4`	Points to a video resource in the sandboxed environment
`jr://file/path/to/file.xml`	Points to an XML resource in the sandboxed environment
`jr://file-csv/path/to/file.csv`	Points to an CSV resource in the sandboxed environment

Virtual Endpoints

“Virtual” refers to the fact that there may or may not be an actual XML document behind the scenes. The URI is resolved locally in any way the client desires. The following are currently supported:

URI format	description
`jr://instance/last-saved`	Refers to the form instance that was **saved** most recently (as opposed to last-opened or last-finalized, for example). The most common use-case for this feature is to “auto-fill” specific form fields with the last-saved value via `odk-instance-first-load`.

Submission

The optional `<submission>` element provides instructions to the client about special submission behavior. The element is placed as a sibling of the primary instance inside the model.

Note that submission behavior can be highly variable between different clients. A client could be 100% ODK-

Forms-spec-compliant but have a custom way of dealing with submissions to fit into an existing system. It is nevertheless considered helpful to document some special behavior that clients may choose to adopt.

```
<model>
    <instance>
        <data id="mysurvey" orx:version="2014083101">
          ...
        </data>
    </instance>
    <submission orx:auto-send="true" />
    <bind nodeset="/data/firstname" type="xsd:string" required="true()" />
    ...
</model>
```

Submission Attributes

The following attributes are supported on the submission element.

attribute	description
`action`	This attribute is optional and can be used to specify a custom URL to send submissions to.
`method`	This attribute is only required and used if the `action` attribute is used. Otherwise it’s ignored. The value should be set to `post`. In the past, the value `form-data-post` was used. Though this is now deprecated, it is recommended that a server accepts submissions for both methods and considers `form-data-post` an alias for `post`.
`base64RsaPublicKey`	This attribute is required to enable encryption. It is a base64-encoded RSA public key. The corresponding private key will be needed to decrypt submissions (and should not be included in the form definition).
`orx:auto-send`	Optional attribute that is either `"false"` or `"true"`. If true, any final records will be sent automatically by the client as soon as a connection is available.
`orx:auto-delete`	Optional attribute that is either `"false"` or `"true"`. If true, and successfully submitted records will be immediately deleted from the client.

Encryption

Forms can enable encryption to provide a mechanism to keep finalized data private even when using **http:** for communications (e.g., when SSL certificate is not there). It provides security for the duration in which the data is stored on a device and on the server.

Encrypted form definitions must have an explicit `<submission/>` element with a `base64RsaPublicKey` attribute.

The client generates a different symmetric encryption key for each finalized form and uses it to encrypt the submission and all media files. The `base64RsaPublicKey` is used to encrypt the symmetric key which is then passed back in a submission manifest.

Here is an excerpt used in an encryption-enabled XForm:

```
<instance>
      <sample id="sample-v1.0">
        <orx:meta>
           <orx:instanceID/>
        </orx:meta>
        <name/>
      </sample>
</instance>
<submission base64RsaPublicKey="MIIBIjANB...JCwIDAQAB"/>
```

Full details on the encryption algorithms and submission manifest can be found here.

Compact Record Representation (for SMS)

ODK XForms records are generally represented as XML using the structure of the primary instance. It is also possible to define how a record can be represented more compactly, usually for SMS submission.

For this representation:

- The value of the `prefix` attribute on the primary instance’s single child is included at the beginning of every record.

- Questions that have a `tag` attribute are represented as the `tag` value followed by the element's value. Questions without a `tag` attribute are omitted.
- The value of the `delimiter` attribute on the primary instance's single child is used to separate components of the compact representation (prefix, tags, values). Defaults to a single space (` `) if not explicitly specified.

Given the following ODK XForm definition:

```
<instance>
    <household id="household_survey" orx:version="2018061801" odk:prefix="hh" odk:delimiter
        <meta>
          <instanceID odk:tag="id" />
        </meta>
        <person>
            <firstname odk:tag="fn" />
            <lastname odk:tag="ln" />
            <age />
        </person>
    </household>
</instance>
```

Full records might look like:

```
<household id="household_survey" orx:version="2018061801" odk:prefix="hh" odk:delimiter="+"
        <meta>
                <instanceID>uuid:82724cc5-df6f-46bf-86d5-26683ae35d5b</instanceID>
        </meta>
        <person>
                <firstname odk:tag="fn" />
                <lastname odk:tag="ln">Bar</lastname>
                <age>10</age>
        </person>
</household>
```

```
<household id="household_survey" orx:version="2018061801" odk:prefix="hh" odk:delimiter="+"
        <meta>
                <instanceID>uuid:82724cc5-df6f-46bf-86d5-26683ae35d5b</instanceID>
        </meta>
        <person>
                <firstname odk:tag="fn">Mary Kate</firstname>
                <lastname odk:tag="ln">Doe</lastname>
                <age>15</age>
        </person>
</household>
```

The compact representations of those records would be: `hh+ln+Bar`

`hh+fn+Mary Kate+ln+Doe`

If the delimiter is included in one of the question values, it will be prepended by a slash. For example, the first name `"Mary Kate"` would be represented as `"Mary\ Kate"` if the default space delimiter is used.

As in the regular representation, nodes that are not relevant are not included in the compact representation. Unlike in the regular representation, nodes that are relevant but empty are not included in the compact representation, even if they have an `odk:tag`.

Future

See the outstanding issues list to get an idea of how this specification will evolve. Join the conversation!