CommCare 2.0 XForms Introduction

CommCare 2.0 introduces a lot of changes to the way that XForms interact with CommCare data. This page isn’t meant to outline all of those changes, but rather to offer some tips for specific tasks that are commonly needed. You can find the full CommCare 2.0 specifications at: https://github.com/dimagi/commcare/wiki/CommCare20Specs

Referencing Case Values

In a form uploaded to HQ as part of the normal app builder, the general template for referencing a case value is of the form (in an XPath Expression)

```
instance('casedb')/casedb/case[@case_id=instance('commcaresession')/session/data/case_id]/VALUE
```

Where VALUE is equal to a named case property, or one of the referenced case properties from the abstract casedb instance, who's structure is located at https://bitbucket.org/commcare/commcare/wiki/casedb.

So to reference the value site_id which was assigned to a case, the reference would be

```
instance('casedb')/casedb/case[@case_id=instance('commcaresession')/session/data/case_id]/site_id
```

and to reference the case's type, you would use the attribute of the case block

```
instance('casedb')/casedb/case[@case_id=instance('commcaresession')/session/data/case_id]/@case_type
```

Note: if your form doesn't already access the casedb (IE load case properties into the form) then you will need to manually instantiate the casedb instance to reference in the form. You can do this by adding the following line in your <model> block (at the same level as the binds) in the source XML:

```
<instance id='casedb' src='jr://instance/casedb'/>
```

Dynamic Multiselect Inputs (Custom Forms Only)

In CC2.0, it is fairly straightforward to use itemsets to control the choices available in selections. Doing so requires setting up four things:

1) Selection Data

Inside your data instance you will need to configure an XML element for each of the options which will be available to select. The general pattern is (all xml element names chosen arbitrarily)

```
<instance>
  <data ...>
    ...
    <q1_value/>
    <q1_items>
      <item id='q1_one'>yes</item>
      <item id='q1_two'>yes</item>
      <item id='q1_three'>yes</item>
      <item id='q1_four'>yes</item>
    </q1_items>
  </data>
</instance>
```

This defines four items for a select question, who's values are specified by the id field. We will use the value of the element itself to control whether the item is displayed as an option to the user. The value will be stored in q1_value

2) Binds

Now we configure the binds which specify which options can be chosen.

```
<bind nodeset='/data/q1_items/item[@id='q1_one']' calculate='if(some_expression, 'yes','no')'/>
<bind nodeset='/data/q1_items/item[@id='q1_three']' calculate='if(another_expression, 'yes','no')'/>
```

These two binds specify logic for the first and third options in the list. It identifies that item one should only be available for selection if some_expression is true, and item three if another_expression is true. In this example, questions two and four are always asked, since their value is set to yes by default and isn't controlled by a calculation.

3) Translations

We'll be using the id's of the items as the direct itext translation key for simplicity. In your itext section, include
4) Defining the Question

Now we simply tie everything together into a select question.

The `<itemset>` specifies that you want to choose from a list of elements in the data model. The `<nodeset>` defines the path to each of those items. The predicate expression `[. = 'yes']` specifies that you only want to include items whose value (the `.`) is equal to `yes`. The `<label>` and `<value>` are both relative to each item in the nodeset, and the label is indexed into the iText.