

THE MM XML FORM BUILDER LANGUAGE V.1.1

An XML Form is an .xml file that consists of pages, groups, items, and relationships between them. The figure below shows the basic structure of an .xml form:



The names of the pages, groups and items have to be a **unique** identifier within the .xml file. If the attribute tag `<submitName>` is not present, this name will be the one used when forming the submission string. `<submitName>` can be used with most fields and it has precedence over the name attribute when it comes to the formation of the submission string.

Pages have been given a meaning that corresponds to that of .pdf receipt pages: We must start and end the `<page></page>` tag making sure we encompass all items that render on a single .pdf page within these tags. Remember that a form Page is a collection of Groups, whereas a .pdf Page is a collection of all the items that render on the same .pdf Page.

GENERALITIES

Each item tag has attributes and tags applied to that item. Any tag in itself can have one or more attributes applied to it:



The following item **tags** are available to all items:

- `<size></size>`,
- `<onmissing><message></message></onmissing>`
- `<label></label>`
- `<detailLabel></detailLabel>`
- `<inlineLabel></inlineLabel>`
- `<position></position>`
- `<placeholder></placeholder>`

dateField will ignore the `<submitName>` tag when it submits its value as three (or two) separate fields as in:

05:birthDateDay:11:birthDateMonth:1978:birthDateYear, where birthDate is the name of the dateField. Use the tag `<prefix>` to specify a different submission key. dateFields also support the `<posfix>` tag.

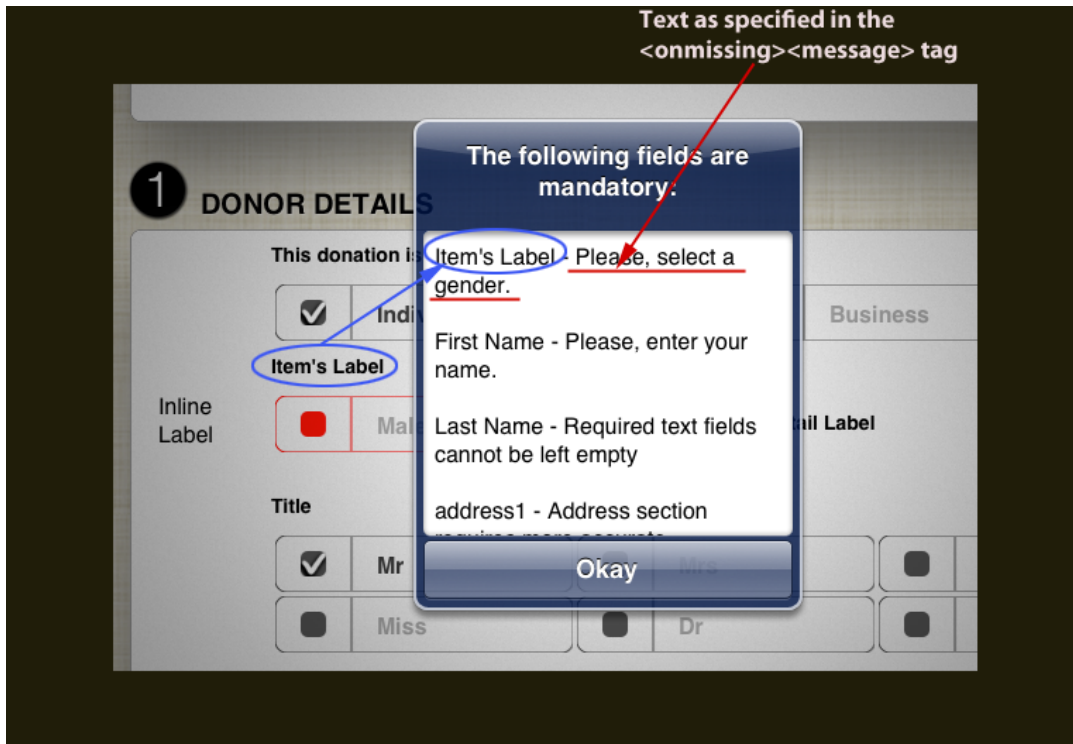
For example, `<prefix>aPrefix</prefix>` applied to the birthDate dateField will cause the submission string to form as:

05:aPrefixDay:11:aPrefixMonth:1978:aPrefixYear.

`<submitName>` works normally on dateFields when these dateFields submit their value as unix time-stamps (by applying the `<unix>YES</unix>` tag) or when submitting their values with specific formats (by applying the `<string> MM-dd-yyyy</string>` tag. Here, the "MM-dd-yyyy" string can be replaced with any valid iOS NSDateFormatter string).

<size>: Specifies the width in pixels of the item when drawn in the Group View. Typically, a value of 620.0 is given for this tag, but smaller values also occur often.

<onmissing><message>: This tag is used to specify the string message that a user sees upon submission when a required item has not been filled up. There are default missing messages, but you can override them by including this item tag.



<position>: This tag specifies the place where the item is drawn within the 9-grid Group View. Its value can be one of the values of the following enumeration:

```
enum {
    center = 0,           //This is the default position within the 9-slice grid.
    upper_left = 1,
    upper_center = 2,
    upper_right = 3,
    center_right = 4,
    bottom_right = 5,
    bottom_center = 6,
    bottom_left = 7,
    center_left = 8,
};
```

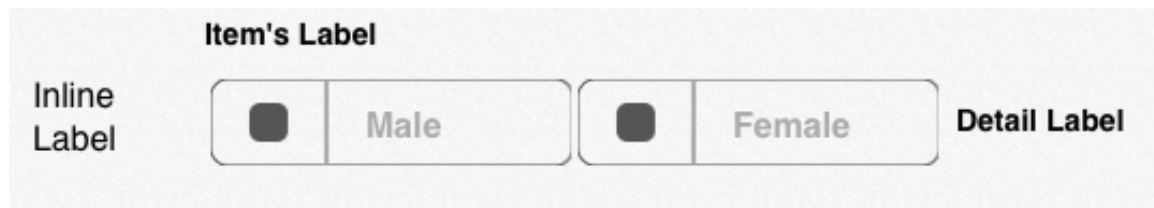
(See *"How the 9-Grid Group View is Drawn"* for more details).

Example:

```

<item name="gender" type="radioControl" multiple="NO" required="YES">
  <size>320</size>
  <columns>2</columns>
  <tag value="Male">Male</tag>
  <tag value="Female">Female</tag>
  <label language="EN_US">Item's Label</label>
  <detailLabel>Detail Label</detailLabel>
  <inlineLabel>Inline Label</inlineLabel>
  <onmissing>
    <message language="EN_US">Please, select a gender.</message>
  </onmissing>
</item>

```

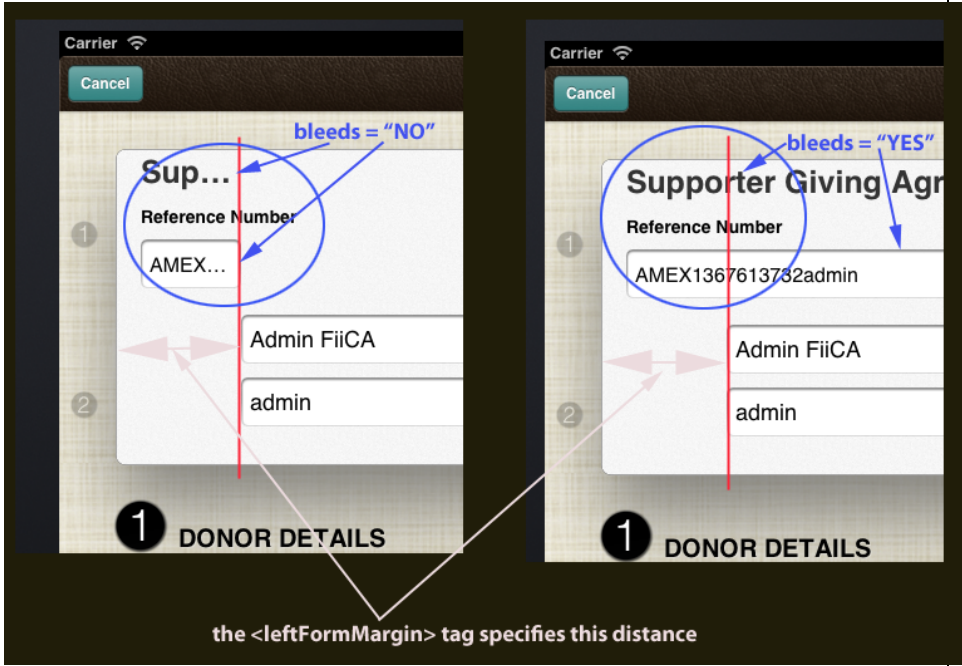
Renders as:

<placeholder>: This is a special tag available to all item types, but that it only displays for textFields and emailPingers in the expected way. For all other items, it can be used to pass along an additional information string to the item.

AVAILABLE ITEM TYPES

iOS FormItem Type:	Corresponding XMLFORM type attribute:
ItemTypeRemoteImage,	remoteImage,
ItemTypeLocalImage,	localImage,
ItemTypeZipImage,	zipImage,
ItemTypeDateField,	dateField,
ItemTypeLabel,	label,
ItemTypeCheckbox,	checkbox,
ItemTypeSegmentedControl,	segmentedControl,
ItemTypeDropDown,	dropDown,
ItemTypeTextField,	textField,
ItemTypeMultilineLabel,	multilinelabel,
ItemTypeRadioSelect,	radioControl,
ItemTypeSpacerBlock,	spacerBlock,
ItemTypeSignatureBox,	signatureBox,
ItemTypePhotoCapture,	photoCapture,
ItemTypeAddressField,	addressField,
ItemTypeEmailPinger,	emailPinger,
ItemTypeCanadianPostalCode,	canadianPostalCode,
ItemTypeValuePicker,	valuePicker,
ItemTypeImagePicker,	imagePicker,
ItemTypeInfoButton,	infoButton,
ItemTypeUKBankPinger,	UKBankPinger,
ItemTypeFRBankPinger	FRBankPinger

AVAILABLE ITEM ATTRIBUTES

XMLFORM item attribute	
name	Mandatory. Must be unique among items
type	Mandatory. One of the FORM types described above
submitAlerts	Optional. Defaults to NO. Upon submission, if a field is left blank, it will bring up an alert asking the user whether they want to continue without filling up that field.
mustValidate	emailPinger only. Defaults to YES. When set to YES, e-mail must be validated in order to submit.
required	When set to YES, the form won't submit if the item is left blank and/or unselected and/or unchanged. For textFields, this attribute paints them peach (or blue if set to NO). Defaults to NO.
bleeds	<p>Defaults to YES. Each item is drawn on one of the 9 elements of the 9-grid view that constitutes a Group View. When the attribute is set to NO, and the item is for example drawn on the upper_left position on the 9-grid, this item will clip its view so that it doesn't bleed into the adjacent section, upper_center. This is almost never the desired behavior.</p>  <p>the <leftFormMargin> tag specifies this distance</p>
isHidden	Defaults to NO. When set to YES, the item is simply not visible, and if the forceSubmit attribute is not set to YES, a hidden fill will not submit its value.
password	textField only. Defaults to NO. When set to YES, masks the input characters as password fields do.
skipSubmit	Defaults to NO. When set to YES, this item will not

	submit its value even if it is visible.
forceSubmit	Defaults to NO. When set to YES, this item will submit its value even if it is hidden.
editable	textField only. Defaults to YES. When set to NO, the textField view renders in white and prevents the user from changing its value.
forceReceipt	Defaults to NO. When set to YES, this item will render on the .pdf whereas the isHidden attribute is set to YES or not.
receipt	<p>When the form attribute pdfType="custom" is present, the only accepted value for this attribute is receipt="other", which will cause new pages to be created on the .pdf with all item having receipt="other". photoCapture is an example of this.</p> <p>When using Made Media .pdf Lily's design, this attribute takes one of the following predefined enumeration values:</p> <ul style="list-style-type: none"> 0= header, 1= map, 2= bannerHeader, 3= banner or bannerView, 4= bannerFooter, 5= personal, 6= payment, 7= other, 8= legal, 9= extra, <p>For example, receipt="map", will cause an item to be drawn on that specific section on the .pdf receipt</p>

When an item isHidden, the skipSubmit attribute has no effect and the forceSubmit attribute has precedence.

When an item is not Hidden, the forceSubmit attribute has no effect and the skipSubmit attribute has precedence

DESCRIPTION OF ALL ITEM TYPES

remoteImage: Currently not active. Forms at this moment don't load images unless they come as part of the .zip package.

Available tags: <path>,<width>,<height>

localImage: Use this item to display an image resource from the Application's bundle on the form. The exact name of the resource must be specified within the <path> tag. List of currently available resources:

loginbrand.png,
cvv_2.jpg,
cheque.jpg,
chequeUS.jpg
cvv_2_es.jpg
cvv_3.jpg
cvv_3_es.jpg
WhiteCircle.png
GreenMark.png
GrayLine.png, etc.

Available tags: <path>,<width>,<height>

zipImage: Use this item type to add images to the form view. Include the _zip.png or _zip.jpg images into the .zip package. As you can see, image names must finish with _zip. Specify the name within the <path> tag.

Available tags: <path>,<width>,<height>

dateField: This field uses the following specific tags:

<prefix>,<formatting>,<defaultDate>,<unix>,<string>,<minAgeRestriction>,<maxAgeRestriction>,<minYearRestriction>,<maxYearRestriction>,<buttonPosition>

<prefix>: A string used to form the submission string. For example, if <prefix>example</prefix> is set and January 05,2013 is selected, the submission string will form as 05:exampleDay:01:exampleMonth:2013:exampleYear, provided not the <unix> nor the <string> tags are set.

<posfix>: Similar to prefix. For example, if `<posfix>Example</posfix>` is set and January 05,2013 is selected, the submission string will form as `05:dayOfExample:01:monthOfExample:2013:yearOfExample`, provided not the `<unix>` nor the `<string>` tags are set.

`<posfix>` and `<prefix>` cannot be used at the same time. When both tags are present, only the posfix tag will be considered.

<formatting>: A string formed with a sequence of the following characters: d,D,m,M,y,Y,a,A separated by the special character "/". The same character can appear only 1,2, or 4 consecutive times within the string. Examples of valid formats are:

```
<formatting>m/yy/dd</formatting>,  
<formatting>yyyy/MM</formatting>, or  
<formatting>D,mm</formatting>
```

Lowercase sequences and uppercase A will display on the form in Spanish. The order of the elements in the sequence determines the order in which the dateField will be displayed on the form.

<defaultDate>: A string representing a unix timestamp or the word today. It will determine the pre-populated date on the dateField
Examples:

```
<defaultDate>today</defaultDate>,  
<defaultDate>281062800</defaultDate>
```

<unix>: When this tag is set with any value, the submission string will form as a single key-value pair, where the key is the name of the dateField unless the `<submitName>` tag is set. Examples:

```
<unix>hello</unix> will submit as: 281062800:nameOfField  
<unix>house</unix> + <submitName>selectedDate</submitName> will submit  
as: 281062800:selectedDate
```

<string>: When this tag is set, the submission string will form as a single key-value pair, where the key is the name of the dateField unless the `<submitName>` tag is set. The value of the tag itself will determine how the string will be formed as established in the NSDateFormatter guidelines. Example of valid arguments are:

```
<string> EEE, MMM d '@' h:mm a</string>  
<string> yyyy-MM-dd</string>
```

<minAgeRestriction>, **<maxAgeRestriction>**: When set, it limits the date selection tools and the validation of the dateField so that users can only pick a value that lies within the values indicated on this tag. Only strictly positive integers are allowed.

<minYearRestriction>, <maxYearRestriction>: When set, it limits the date selection tools and the validation of the dateField so that users can only pick a value that lies within the values indicated on this tag. Positive, negative and zero values are allowed, Examples:

The pair:

```
<minYearRestriction>-10</minYearRestriction>  
<maxYearRestriction>0</maxYearRestriction>
```

will limit the user to choose a date from 10 years into the past up to today's date, whereas the pair:

```
<minYearRestriction>0</minYearRestriction>  
<maxYearRestriction>+10</maxYearRestriction>
```

will only accept a date from today and 10 years into the future.

<buttonPosition>: One of the elements in the enumeration up left, up right, low right, low left. It determines where the DONE button will show in the date picker. It defaults to the up_left position. Other semantically equivalent keywords are valid. Examples:

```
<buttonPosition>default</buttonPosition>  
<buttonPosition>upperleft</buttonPosition>  
<buttonPosition>up left</buttonPosition>
```

will all be understood to mean
<buttonPosition>up_left</buttonPosition>

label: A single text label. It renders its view according the following tags:

```
<text>,<textColor>,<textSize>
```

<text>: It has the optional "encoding" attribute that for now can only be set to "base64" or "URLEncode". URLEncode is the default value for this attribute. Currently, base64 encoding is the only way we have to render accented and other special characters on the form. Example:

```
<text encoding="base64">Ts06bWVybyBkZSBSZWZlcmVuY2lh0g==</text>, will  
display as "Número de Referencia"
```

<textColor>: Accepts RGB strings such as #FF0000, but also key words such as transparent or clear, red, blue, white, black or dark, orange, green, yellow, brown, gray or grey, and light gray, are understood.

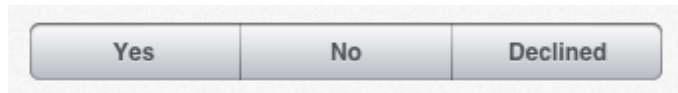
checkbox: This item accepts the `isChecked`, and `submitsValue` attributes. The `submitName` tag is only understood in the most recent version of the XMLCore, so make sure line 193 of the `submissionString` method in the `GroupView.m` file is changed from `anItem.name` to `anItem.submitName`.

`isChecked` and `submitsValue` both default to NO. When `isChecked` is set to YES, the checkbox displays with the box checked initially. When `submitsValue` is set to YES, a 0 or a 1 will be submitted to the backend.

segmentedControl: Displays classic, iOS-like segmented controls. It uses the `<tag>` tag to specify the available selections. Example:

```
<item name="receivedWp2" type="segmentedControl">
    <size>320</size>
    <tag value="Y">Yes</tag>
    <tag value="N">No</tag>
    <tag value="D">Declined</tag>
</item>
```

will display as:



dropDown: It uses the `<option>` tag to specify the available selections. For example:

`<option value="3">03 - Street</option>` will specify one option.

textField: The most used type of item. It accepts the following tags:

`<textColor>`, `<textSize>`, `<minlength>`, `<maxlength>`, `<minValue>`, `<maxValue>`, `<bgcolor>`, `<validation>`, `<keyboardtype>`, `<action>`, `<partClone>`, `<defaultValue>`, `<regex>`, and `<regexmessage>`.

<validation>: One of the elements in the enumeration:

```
typedef enum {
    none,                //No validation performed at all.

    lettersOnly,         //Alphabetic characters in most western languages. No spaces or punctuation. Uses 'ascii' keyboard
    letters,              //Same as lettersOnly + spaces and punctuation. Uses 'ascii' keyboard.
    numericOnly,          //[0-9] No spaces or any other characerts. Uses 'numberPad' keyboard.
    numericWithPeriods,  //Same as numericOnly but periods ARE permitted. Uses 'decimalPad' keyboard.
    numeric,              //[0-9] + spaces, dashes and other punctuation symbols. Uses 'numbersAndPunctuation' keyboard
    lettersNumericOnly,   //Same as lettersOnly + numericOnly. No spaces or punctuation permitted. Uses 'default'
    lettersNumeric,       //Same as letters + numericOnly. Uses 'default' keyboard
    alphaOnly,           //[A-Z], [a-z] only. No spaces, dashes, or any other punctuation permitted. Uses, 'default' keyboard.
```

```

alpha,           //Same as alphaOnly, but spaces, and punctuation permitted. Uses 'default keyboard'
alphanumericOnly, //Same as alphaOnly + numericOnly. No spaces, dashes or apostrophes. Uses 'default'
plain,           //Same as alphaOnly, but accepts spaces.
alphanumeric,    // Same as alpha + numericOnly. Uses 'default' keyboard
intlphonenumber, //numericOnly + parentheses and dashes allowed. Uses 'phonePad' keyboard.
usphonenumber,   //intlphonenumber, but it checks for specific US format. Uses 'phonePad' keyboard.
creditcard,      //It applies Luhn's algorithm for basic validation. Uses 'numbersAndPunctuation' keyboard
email,           //Checks for basic validity. Uses 'emailAddress' keyboard.
day,             //Checks if the number is between 1 and 31. Uses 'numberPad' keyboard.
month,           //Checks if the number is between 1 and 12. Uses 'numberPad' keyboard.
year,            //Checks if the number is between 1900 and 2100. Uses 'numberPad' keyboard.
luhn,            //Chechs against Luhn's algorithm
names,           //Same as letters, but capitalizes every word.
namesNumeric,    //Same as names, but includes numbers.
repeat,          //Special type of validation for repeatFields, like creditCard and repeatCreditCard
}

```

<keyboardtype>: One of the elements in the enumeration:

```

typedef enum {
    defaultKeyboard, //The default keyboard to use. UIKeyboardTypeDefault in iOS.
    asciiKeyboard,   //Keyboard that displays standard ASCII characters. UIKeyboardTypeASCIICapable in iOS.
    numbersAndPunctuation, //UIKeyboardTypeNumbersAndPunctuation
    URLKeyboard,      //UIKeyboardTypeURL, features “.”, “/”, and “.com” prominently.
    numberPad,        //UIKeyboardTypeNumberPad, designed for PIN entry. This type features the numbers 0 through 9.
    phonePad,         //UIKeyboardTypePhonePad, features the numbers 0 through 9 and the “*” and “#” characters.
    namePhonePad,     //UIKeyboardTypeNamePhonePad in iOS.
    emailAddressKeyboard, //UIKeyboardTypeEmailAddress in iOS.
    decimalPad,       //UIKeyboardTypeDecimalPad. Use a keypad with numbers and a decimal point.
}

```

If <keyboardtype> is specifically given, it overrides the default keyboard given based on validation type.

<action>: By default, textFields don’t trigger actions when they are done editing. Use the “itemShouldNotify:” action to trigger an event when a textField loses focus once is done editing:

<action>itemShouldNotify:</action>

This action can be used to check for a textField’s value immediately after finishing editing it. For example, we may want to do this to evaluate a relationship condition right when the textField is done editing.

NOTE: Using the itemShouldNotify: action will cause the present keyboard to hide. This could result in a poor user experience when filling up a long form using the Next keyboard button.

<partClone>: A tag that allows a textField to clone all of part of its value and submit it to the backend with a different name. For example:

```
<partClone>4::newField</partClone>
```

will cause the first 4 characters of the value of the textField to submit to the backend under the key newField. The first parameter is always a positive, negative or zero value. A zero value will copy the entire value, and a negative value, for example of -3, will copy the last three numbers of the item value.

<regex>: Overrides any type of validation given to this textField and substitutes it with the corresponding Regular Expression passed on as an argument of this tag. The syntax of valid regex expressions follow the guidelines as specified in the iOS NSRegularExpression class.

<regexmessage>: This is the error message that will appear in red if the user fails to input a value on the textField that complies to the expression given in <regex> while they input the value. This is different from the <onmissing> tag that all items enforce.

multilineLabel: This is very similar to the label type, but this item type provides more customization than a label. The following tags are accepted by a multilineLabel item in addition to all the tags supported by every item and the tags supported by the label type:

<sizeY>, <numberOfLines>, <align>

where <align> can take one of the values "center", "right", or "left", being left" the default value.

radioControl: accepts the "multiple" attribute, which defaults to NO. When multiple="YES" the radioControl allows us to select multiple options at once. radioControl uses the <tag> tag in the same way segmentedControl does to specify its options.

radioControl supports the following special tags:

<columns>, <override>, <minRequiredItems>, and <maxRequiredItems>

<columns>: Specifies how many buttons are displayed per row in the selection grid.

<override>: For example, `<override value="15">HOUSE</override>`, means that if you have four tags and all of them get selected (value = 15 = 1+2+4+8), the value of the control is then HOUSE instead of 15.

<minRequiredItems>: Returns FALSE upon validation when less than minRequiredItems have been selected.

<maxRequiredItems>: Returns FALSE upon validation when more than maxRequiredItems have been selected.

When minRequiredItems and maxRequiredItems have the same value, user must select precisely this number of items to return TRUE upon validation. When set in multiple mode, radioControl uses binary values to account for its current state value, where the first given tag carries the value 2^0 .

imagePicker: This control behaves exactly as the radioControl, but the available options are specified by using the image tag.

For example,

```
<image value="aValue1">fileName1.png</image>
<image value="aValue2">fileName2.png</image>
```

will display the two images fileName1.png and fileName2.png and will let you select one of them when the attribute multiple="YES" is not present. If the first image is selected, the value aValue1 will be send to the backend upon submission.

spacerBlock: An item whose only purpose is to move items down in the Y coordinate to avoid overlaps and other drawing issues. Spacer blocks are used by specifying two tags:

```
<item name="group5Spacer" type="spacerBlock" required="NO">
  <sizeY>180</sizeY>
  <position>upper_right</position>
</item>
```

Where the value of the <sizeY> tag will push things down 180 pixels on the upper_right section of the group's 9-grid view. See "How the 9-grid Group View is Drawn" for details.

signatureBox: A signature box that behaves in a button-like manner bringing up the signature panel when it's tapped on. It enforces the submitsValue attribute in the same way checkboxes do, submitting to the backend in a key-value form when set to yes, submitting 0 when a signatureBox was not signed and 1 otherwise. It allows for the following special tags: **<width>**, **<height>**

photoCapture: Same as signatureBox, but it brings up the camera UI to capture a picture.

addressField: A composed item type that will display several textFields and dropDowns at once, allowing the user to input a complete address. The following tags are supported by addressFields:

<country>, <validation>, <prefix>, <desiredCountryCode>

<country>: A string that specifies either the US, Canada or Mexico. All reasonable strings are accepted ["ca" was only recently added to the XMLCore; check line 85 of the iOSAddressItem.m file to verify this]. For example,

```
<country> united states</country>
<country> mx</country>
```

are all valid.

<validationService>: At the time when this documentation was written only one validation method is supported, so any value of this tag will default to the QAS validation system. For example, all

```
<validation>QAS</validation>
<validation>THIS</validation>
<validation>POSTCODE</validation>
```

will result in a Validation Button displaying and running QAS in the backend to validate addresses. Omitting the <validationService> tag will suppress the validation button from the form.

<validationService> replaces the previous <validation> tag, but will only work with the most recent releases of the onBoard App. If the use of this tag is not giving the desired behavior, switch back to the old <validation> tag.

<prefix>: This is a string used to form the submission string in a way similar to the dateField when a prefix is set: For example, if <prefix>myHouse</prefix> is set, then the submission string for this address field will be:

```
streetName:myHouseAddress1:Apt.1:myHouseAddress2:Montreal:myHouseCity:H
3T1W5:myHousePostalCode:QC:myHouseStateProv:Canada:myHouseCountry
```

If no prefix is set, addressFields don't use the item name as prefix, which means that the default submission string for any addressField is:

```
streetName:address1:Apt.1:address2:Montreal:city:H3T1W5:postalCode:QC:s
tateProv:Canada:country
```

<desiredCountryCode>: This tag only has an effect for the last part of the submission string. By default, country codes submitted to the backend will be CAN, USA, and MEX. If a different code is needed, for example, CA, one can set the value of this tag to CA:

```
<desiredCountryCode>CA/<desiredCountryCode>
```

emailPinger: Another composite item that will display a textField along with a button enabled to validate e-mail addresses. emailPinger supports the mustValidate special attribute that defaults to YES. When set to NO, the contour of the "Validate Email" button will not highlight red when no validation of the e-mail address has taken place. When set to yes, the "Validate Email" button will be highlighted red, and the form won't submit unless the e-mail address has been verified. The <placeholder> tag was only added to the most recent version of the onBoard app and it is not available in previous versions.

canadianPostalCode: A special textField that supports only the following four tags supported by all textFields:

```
<textColor>,<textSize>,<size>,<bgcolor>
```

This item type displays the desired letter-to-number keyboard switching behaviour.

valuePicker: Another special textField that doesn't allow for editing, but rather brings up a selection panel to choose a value from a list. The list of items is specified using the same "option" paradigm used with dropDowns:

```
<option value="AL">Alabama</option>
```

The following special tags are supported:

```
<options>,<order>,<buttonPosition>
```

<options>: This allows you to specify a range of integer values without having to list each one using <option> tags. For example, <options>10-15</options> will display all values 10,11,12,13,14, and 15 in the picker.

<order>: Let's you specify if you want the picker to order its values in ascending or descending order. "ascending" or "a" and "descending" or "d" are the only valid values for this tag.

<buttonPosition>: Same as the buttonPosition tag for dateFields: One of the elements in the enumeration up left, up right, low right, low left. It determines where the DONE button will show in the date

picker. It defaults to the `up_left` position. Other semantically equivalent keywords are valid. Examples:

```
<buttonPosition>default</buttonPosition>
<buttonPosition>upperleft</buttonPosition>
<buttonPosition>up left</buttonPosition>
```

will all be understood to mean
`<buttonPosition>up_left</buttonPosition>`

infoButton: A poorly designed button that helps to display additional information on the form in a modal-like form.

The actual text on the button is specified using the `<defaultValue>` tag and the actual text detail is given by the `<text>` tag. `<textSize>` is used to specify the size of the text in the modal and not on the control itself.

UKBankPinger: A composite field similar to an `addressField` that displays information relevant when collecting funds in the UK. It works together with the POSTCODE service hosted at

<https://madeportal.com/postcodeEntry/BankInterface.php>

to retrieve the bank's address information given the Sort Code and Account Number. **

FRBankPinger: A composite field similar to the `UKBankPinger` that displays information relevant when collecting funds in France. It works together with the `iban-bic.com` service through the

[bankValidation.php](#)

script to retrieve the bank's address and BIC information given the IBAN number. The service could potentially also return the address of the bank, but this behavior hasn't been present on the field up to now.
**

****ATTENTION:** Since this third-party validation services could return data formatted in many different forms, it is advisable for the backend to take care of for example, special characters such as a comma (,) being submitted. When creating an export file in `.csv` format, for example, the presence of a comma could result in a malformed file.

RELATIONSHIPS

Relationships are if-else-like structures that evaluate a 'condition' statement of the form (itemName OPERATOR selectedValue) and perform a change on the desired object (another form item). The basic structure of a relationship is:

```
<relationship object="type::targetItemName">
  <condition>itemName OPERATOR "US"</condition>
  <if>
    <validTagForObject>value</validTagForObject>
  </if>
  <else>
    <validTagForObject>value</validTagForObject>*
  </else>
</relationship>
```

where **type** can be one of the three words: **item**, **group**, or **page**. **OPERATOR** can be one of: **AND**, **OR**, **BT**, **LT**, **==**, **!=** . It is possible to create complex conditions by using brackets and different itemNames. For example,

```
<condition>citizenship == "US" AND (gender == "M" OR age BT
5)</condition>
```

is a valid condition.

*because of a flaw in the design of the `-(void)modifyTarget:(NSString *)target withElements:(NSDictionary *)elements` function inside the `PageManager.m` file, only ONE tag value can be processed at the time. This means that if-conditions with more than 1 subtags will likely fail to evaluate properly:

```
<if>
  <validTagForObject1>value1</validTagForObject1>
  <validTagForObject2>value2</validTagForObject2>
</if>
```

will probably result in unexpected behavior. Having said this, options for dropDowns and radioControls can be setup at once using this mechanism. For example:

```
<else>
  <tag value="1">Afrique</tag>
  <tag value="2">Inde</tag>
  <tag value="3">Asie du Sud Est</tag>
</else>
```

will setup the options of certain radioControls adequately.

When the object type is a page or a group (`<relationship object="page::pageName">` or `<relationship object="group::groupName">`), the only allowed validTagsForObject are:

`<show>`, and `<hide>*`

*The values for these tags are entirely optional, but they cannot be empty. For example, `<show>THIS</show>` and `<hide>DOG</hide>` are perfectly acceptable values.

When the object type is a item (<relationship object="item::itemName">), every valid tag for that item is allowed. In addition to these tags, the following special tags are also allowed:

<show> and <hide>, <evaluate>, <copy>, <submit> and <skip> or <force>, and <editable>

<show> and <hide>: These tags behave as expected, hiding or showing a particular item in the form when certain condition arises.

<evaluate>: This tag is used to perform operations between items. For example, <evaluate>monthlyIncome + otherIncome</evaluate> will fillup the object textField value with the result of the addition between the current values of the monthlyIncome and the otherIncome textFields. At the moment this documentation was written, only the following three operations are allowed:

- Concatenation of two textFields using any character as connector:

Example:

If textField1 has value "value1" and textField2 has value "value2",

<evaluate>textField1.- textField2</evaluate> will populate the target object textField with the value "value1-value2".

<evaluate>textField1./textField2</evaluate> will populate the target object textField with the value "value1/value2".

A single space " " after the concatenation operator will concatenate both values together by means of an empty connector:

<evaluate>textField1. textField2</evaluate> will populate the target object textField with the value "value1value2".

If you want to concatenate two textField using a space as a connector, use a **double space** after the concatenation operator:

<evaluate>textField1. textField2</evaluate> will populate the target object textField with the value "value1 value2".

- Addition or subtraction of two textFields:

Attention: Attempting to add two textFields that do not have a string value that can be interpreted as an integer value will result in a crash!

Example:

If textField1 has value "5" and textField2 has value "7",

`<evaluate>textField1+textField2</evaluate>` will populate the target object `textField` with the value "12".

`<evaluate>textField1-textField2</evaluate>` will populate the target object `textField` with the value "-2".

- Addition or subtraction of certain number of milliseconds to a `dateField` target Object

Example:

`<evaluate>+655565</evaluate>` will populate the target `dateField` object with a date 655565 seconds in the future with respect of the current date.

<copy>: This tag copies the current value of a `textField`, `valuePicker`, `canadianPostalCode` or `addressField` item into a `textField`, `valuePicker`, or `canadianPostalCode` target objects.

When the value to copy is an `addressField`, the following syntax is understood:

`<copy>addressToCopy::address1,address2,address3,city</copy>`. Here, the merge of the `address1`, `address2`, `address3` and `city` fields from the `addressToCopy` `addressField` will be merged and copied into a `textField`, `valuePicker`, or `canadianPostalCode` as specified by the object target field. (Remember that **addressFields** have the following 7 predefined components to choose from:
address1, address2, address3, city, postalCode, stateProv, and country)

When the value to copy is an `FRBankPinger`, the following syntax is understood:

`<copy>bankToCopy:: name,address,postCode,city</copy>`. Here, the merge of the `name`, `address`, `postCode`, and `city` fields from the `bankToCopy` `FRBankPinger` will be merged and copied into a `textField`, `valuePicker`, or `canadianPostalCode` as specified by the object target field. (Remember that **FRBankPinger** have the following 10 predefined components to choose from:
name, address, postCode, city, sort, agence, account, rib, iban, bic)

for **UKBankPingers**, the same applies, but they only have 7 predefined components: **name, address, postCode, name1, name2, sort, account**

Examples:

`<copy>textField1</copy>` will copy current `textField1`'s value into the object Target, but `<copy>addressField1</copy>` won't do anything.

`<copy>textField1,textField2</copy>` will copy current `textField1`'s value, concatenated with current `textField2`'s value by means of a ". " (dot+space) into the object Target.

copy operations have been extended to support the following source-target combinations:

Source:	Target:
UKBankPinger FRBankPinger addressField textField valuePicker canadianPostalCode radioControl imagePicker	textField, valuePicker, canadianPostalCode
addressField textField dropDown imagePicker radioControl	dropDown
addressField	addressField

<submit>: <submit>THIS</submit> sets the skipSubmit attribute on an item to FALSE: `thisItem.skipSubmit = NO`. This tag is being deprecated and replaced by the tag <force>

<skip>: <skip>THIS</skip> sets the skipSubmit attribute on an item to TRUE (`thisItem.skipSubmit = YES`), and the forceSubmit attribute of the same item to FALSE (`thisItem.forceSubmit = NO`).

<force>: <force>THIS</force> sets the skipSubmit attribute on an item to FALSE (`thisItem.skipSubmit = NO`), and the forceSubmit attribute of the same item to TRUE (`thisItem.forceSubmit = YES`).

<skip> and <force> form a mutually exclusive pair! When skipSubmit is set to TRUE, the object target item will not submit to the backend. In the same way, if the forceSubmit property is set to TRUE, this item will submit its value to the backend whether the item is hidden or not.

OTHER FEATURES

LOCALIZATION: Every tag that displays text can be localized by means of the language attribute. All `<text>`, `<label>`, `<inlineLabel>`, `<detailLabel>`, `<message>`, `<placeholder>`, `<tag>`, and `<regexmessage>` tags can be easily localized directly on the XML file by specifying the language using the 5-character language code as described in `NSLocale.availableLocaleIdentifiers`. Possible language-attribute examples are:

```
<message language="EN_US">Please, select a gender.</message>
<message language="EN_UK">Please, select your sex.</message>
<message language="ES_MX">Por favor, digame su sexo.</message>
```

Recall that accented characters can only be interpreted properly by adding the optional encoding attribute.

ACTIONS

Actions are hardcoded functions that can be invoked upon a change of one specific item. By default, dateFields, checkboxes, segmentedControls, dropDowns, and radioSelectors always notify the Application when they change their state, and then a search is run through all relationships to determine if such change has an impact on other items. A change in a textfield, in general, doesn't trigger such a search, but it can be triggered by means of an action tag:

`<action> itemShouldNotify:</action>`

This action will notify the App that we wish to look for a relationship when a particular textfield or valuePicker finishes editing.

Other 2 actions have been hardcoded in the MobiPaper app:

`<action>linksTo::certainCreditCardTypeField</action>` (Used for credit cards).

Example:

```
<item name="_creditCardType" type="dropDown" required="YES">
  <option value="Visa">Visa</option>
  <option value="Mastercard">MasterCard</option>
  <option value="AmericanExpress">American Express</option>
  <option value="DiscoverNovus">Discover</option>
  <size>620</size>
  <textSize>26.0</textSize>
  <label>Card Type:</label>
</item>
<item name="_creditCardNumber" type="textField" required="YES">
  <size>620</size>
  <textSize>26.0</textSize>
  <placeholder>Card Number</placeholder>
  <validation>creditcard</validation>
  <action>linksTo::_creditCardType</action>
</item>
```

This tells the `_creditCardNumber` field to first check if the `_creditCardType` has been selected first before it allows you to input anything on this field. For this action to work, both linked items must be part of the same group. If the action tag was missing, the user could go ahead and type something on the `_creditCardNumber` field before selecting a credit card type.

The previous two examples show the flexibility of the action tag. It can trigger predefined behavior, before validation, after validation, before the value of an item has changed or after a change, depending how the predefined behavior was coded.

The following action shows another use of the action tag: In addition to notifying the App that an item's value has changed, checkboxes, segmentedControls, and dropDowns may want to trigger another action after their value changed. At the moment this documentation was written dateFields and radioSelect tools don't support the action tag.

<action>UFUcreateSourceCode</action> (Specific function for our UFU Client)

This action will accept a change on the field that includes this tag, and immediately after it will call the hardcoded `UFUcreateSourceCode` function defined in the `GroupView.m` file to update a field on the form with a complex value that is calculated based on the recent change and on the value of other items on the form. This complex calculation was too hard to be supported natively on the `.xml` file, and this is why it had to be hardcoded in the App itself.

<action>SumItems</action> This action was created to support certain arithmetic operations for the sales App, and it is used together with the **<actionArg>** tag. Pivotal is an example of how this action works:

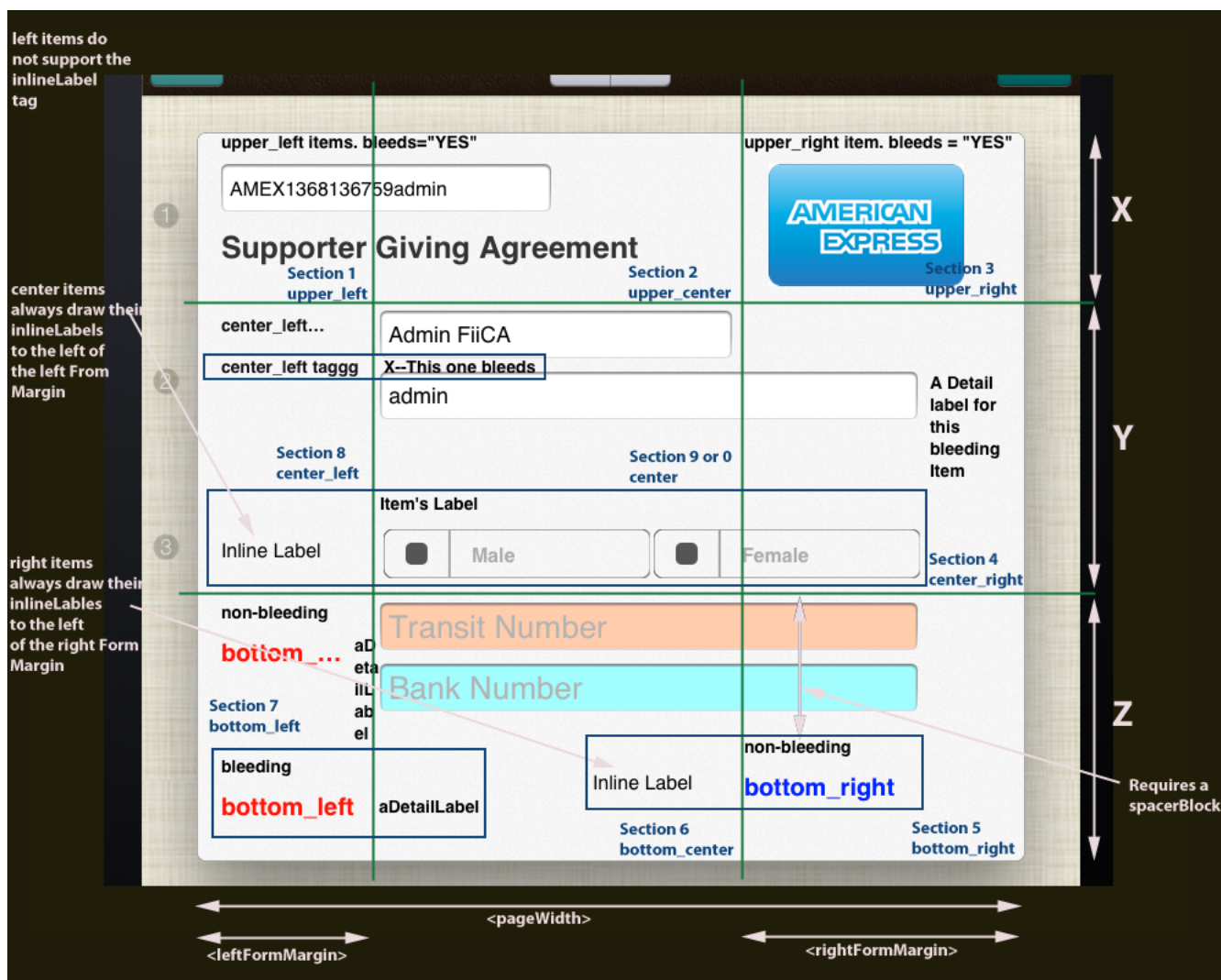
```
<action>SumItems:</action>
<actionArg>MAXMIN::100,50::owner1Percent,owner2Percent</actionArg>
```

The two fields `owner1Percent` and `owner2Percent` must add up to a maximum value of 100 and a minimum value of 50, when this is not the case, the fields will be highlighted and the form cannot be submitted.

```
<action>SumItems:</action>
<actionArg>EQ::100::cardSwipe,keyed::YES</actionArg>
```

In this case, the two fields `cardSwipe` and `keyed` must add up to EXACTLY a value of 100. When the 4th `actionArg` argument is set to YES, the second field not being filled up on the form will automatically fill to add up to a 100 as soon as the first field being filled up resigns focus. If the 4th argument is not present or set to NO, the fields will highlight and the form won't submit until proper values adding up to 100 are given.

HOW THE 9-GRID GROUP VIEW IS DRAWN



The 9-grid Group View consists of the following sections:

```
typedef enum {
    center,          //This is the default position within the 9-slice grid.
    upper_left,
    upper_center,
    upper_right,
    center_right,
    bottom_right,
    bottom_center,
    bottom_left,
    center_left,
} ItemPosition;
```

The widths of each of these sections are determined by the form tags `<leftFormMargin>`, `<rightFormMargin>` and `<pageWidth>`. Notice that the preferred value for `<pageWidth>` is always 678.0, but this could be changed.


The heights of each section depend on the number of items being drawn on that section. Items draw one underneath the other ones in a row-like form.

The value on X on the image above, is determined by the maximum height among sections 1, 2, and 3. Y is the maximum height of sections 8, 9, and 4, and $Z = \max(\text{sec 7}, \text{sec 6}, \text{sec 5})$. Upper items always draw above the first horizontal green line. Center items in between the two horizontal lines, and bottom items draw under the second horizontal line.

Notice that when items on a section are laid out, they start drawing at the top of that section. In the image above, this means that if we had drawn the `bottom_right` label in Section 5 without the use of a `spacerBlock` item, this label would have overlapped with the peach and blue textFields in Section 6, which have been given an attribute `bleeds="YES"`.

Remember that by default, all items are given an attribute `bleeds="YES"`. If you wish to constrain an element to its section only, you have to set `bleeds="NO"` for that item, like the first `center_left` label tag in Section 8 or the non-bleeding `bottom_left` tag in Section 7 in the image above. Both of these items have been given an attribute `bleeds="NO"`, which is why they truncate their text not to bleed onto the adjacent sections.

<inlineLabel>, <label>, and <detailLabel>:



The diagram illustrates the layout of an item with labels. At the top, the text "Item's Label" is centered. Below it, there are three main components: "Inline Label", a central container, and "Detail Label". The "Inline Label" is on the left. The central container is divided into two parts: a left part with a black square icon and the word "Male", and a right part with a black square icon and the word "Female". The "Detail Label" is on the right.

When an item is added to the .xml form, it can always be given any or all of the three tags mentioned above.

The <label> Item's Label always draws slightly above the item itself, with black, bold letters.

The <inlineLabel> Inline Label always draws on the left of the item itself. <inlineLabel> is ignored by all left items (items drawn on sections 1, 8, or 7), and Section 9 items always draw their inline labels on the adjacent left section (Section 8). In a similar way, right items (items on sections 3, 4, or 5) always draw their inline labels on the corresponding left adjacent section (on the left of the second vertical line).

The <detailLabel> Detail Label always draws to the right of the item with black, bold letters, and it expands downwards to a maximum of 10 lines to accommodate all its text. If the total amount of text doesn't fit in those 10 lines, the text truncates then. Notice that the detailLabel could cause the vertical size of an entire section to increase, like the detailLabel on the second, non-editable, bleeding textField on the center section, which causes the second horizontal green line to move shift down.