# Using regular expressions

## Regular Expressions

A regular expression is a special text string for describing a search pattern. Regular expressions find an important use in specifying constraints in your forms. You might want to set constraints on the length, character set allowed and various other fields in the data input by the user. All this can be achieved by using regular expression constraints in your form.

Some basic operations on regular expressions are:

1. **Boolean or**: A vertical bar separates alternatives. For example, `gray|grey` can match gray or grey.
2. **Grouping Parentheses**: They are used to define the scope and precedence of the operators. For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of gray or grey.
3. **Quantification**: A quantifier after a token (such as a character) or group specifies how often that preceding element is allowed to occur.

    > - Square brackets indicate the occurrence of elements within. For example, `[abc]` matches either a, b or c character, `[a-g]` matches any character from a to g, `[^abc]` matches any character except a,b or c.
    > - The question mark indicates zero or one occurrence of the preceding element. For example, `colou?r` matches both color and colour.
    > - The asterisk indicates zero or more occurrences of the preceding element. For example, `ab*c` matches ac, abc, abbc, abbbc, and so on.
    > - The plus sign indicates one or more occurrences of the preceding element. For example, `ab+c` matches abc, abbc, abbbc, and so on, but not ac.
    > - `{n}`: The preceding item is matched exactly n times.
    > - `{min,}`: The preceding item is matched min or more times.
    > - `{min,max}`: The preceding item is matched at least min times, but not more than max times.

For a clear understanding of regular expressions, try these regex online checker tools:

- https://regex101.com/
- https://www.regextester.com/
- https://regexr.com/
- http://www.regexe.com/.

## Tips on using regular expressions

- Regular expressions only apply to **strings**. The function used is of the form: `regex(string value, string expression)` where it returns the result of regex test on provided value. If the result is true, the input value is valid and satisfies the constraint. If the result is false, input value is not valid and user may need to re-enter the input. The regex function will be placed in the constraints column in your XLSForm.
- Be careful while setting limits on length of a constraint. For example, if you want to restrict a text input to a maximum of six alphabetic characters, you can do so as follows:

In **XLSForm**:

**Survey**

| type | name | label | constraint | constraint_message |
|---|---|---|---|---|
| text | ex | Enter example | regex(.,'^[a-zA-Z]{0,6}$') | Input can have a maximum of 6 alphabetic characters. |

In **XForm XML**:

```
<bind constraint="regex(.,'^[a-zA-Z]{0,6}$')" jr:constraintMsg="Input can have a maximum of 6 alphabetic characters."

<input ref="/regex_ex/ex">
   <label>Enter example</label>
 </input>
```

Instead if you used a constraint of the form `regex(.,'^[a-zA-Z]{6}$')`, it would require an input of exactly six alphabetic characters.

Note

- `.` denotes the input string.
- `^` and `$` denote start and end of string respectively.

- If you want to use a regular expression constraint on a number, make sure that the type of your question is **text** and appearance is **numbers** and then apply the constraint.

The following example will validate a 10-digit North American telephone number. Separators are not required, but can include spaces, hyphens, or periods. Parentheses around the area code are also optional.

In **XLSForm**:

**Survey**

| type | name | label | constraint | constraint_message | appearance |
|---|---|---|---|---|---|
| text | tel_no | Enter your Telephone number | regex(.,'^(([0-9]{1})*[- .(]*([0-9]{3})[- .)]*[0-9]{3}[- .]*[0-9]{4})+$') | Telephone numbers should have 10 digits with optional separators. | numbers |

In **XForm XML**:

```
<bind constraint="regex(.,'^(([0-9]{1})*[- .(]*([0-9]{3})[- .)]*[0-9]{3}[- .]*[0-9]{4})+$')" jr:constraintMsg="Teleph

<input appearance="numbers" ref="/regex_ex/tel_no">
   <label>Enter your Telephone number"</label>
</input>
```

An other alternative to this would be to use a regular expression of the form: `regex(string(.),'...')`. But this should be avoided since the value of *string(.)* would be after whatever you entered was converted to an integer. So if you entered 0004, string(.) would be 4.

- Integers are limited by binary representation to 9 decimal digits. If you want something longer (like 10 numbers) then make sure to use a text type with appearance as numbers and add a constraint restricting the input string to be a number. Constraint is required since appearance setting changes the keyboard style of the pop-up keyboard to the number keyboard when you attempt to enter data into the field but does not prevent non-numbers from being entered. This relies upon the device's keyboard supporting (See this).

For example, a constraint of the form `regex(.,'^[0-9]{11}$')` will restrict the input string to be a number of exactly 11 digits.

- Avoid using complex regex patterns as that may cause stack overflow crashes. Also, avoid placing constraints on names since your regex will certainly not capture all the punctuation or random characters that names can contain and they are hard error-prone and hard to maintain.

**See also**

You can refer this list for various common regex patterns.