XML Editor

This is a prototype editor for the XML files. It converts XML into a tree structure. The tree structure has various class types that provides different properties for the various parts of the tree, Sections, Question Groups, Questions, etc. These objects have a BaseTreeNode superclass which is used in the admin views. Admin change-lists normally work with a single model type and in this case it is the BaseTreeNode model we use to set up the admin interface. All our objects are of this type.

The admin views allow copy and paste of tree objects and also drag and drop.

There are 3 management functions described in the Usage section which can import and export xml. It is also possible to render a questionnaire's XML in the browser. An additional feature is the `buildapp` command which will create a basic Django app and build models to match the question in the XML and a basic admin set up as well. It was not possible to build Django forms automatically at this stage because TextNode labels have no connection to Questions. If TextNode were connected to Questions it would be possible to build Django Forms automatically. These forms can render their own HTML automatically which could be useful in some cases. They can also provide validation.

Issues

In building this structure, some identifier had to be used so that the tree objects have an indentifier of some kind in the admin interface. I have used a title for this but many things in the XML don't have titles. I have in some cases put in a title so that the objects can be used in the Admin interface. The problem being that exported XML has this false title in it. This would require some decisions to fix about what the identifier should be and how it would be stripped out of the output if it shouldn't be there. The importxml functions and classes are in the management/commands folder and subclass the XML Objectifier's class. This subclass provides a few functions to convert the XML Objectifiers objects into the new tree structure objects described in this project's models.py.

Installation

Note: This commands below are dependent on Django 1.7 at the moment. 1.8 has changed the argument parsing so that will need fixing.

Install the objectifier dependency.

```
pip install git+git://github.com/cam-mrc-epid/xml-objectifier

OR

pip install
svn+https://svn.mrc-epid.cam.ac.uk/private/MRC_Webforms_Python/xml-objectifier/
```

Install the Editor app.

```
pip install git+git://github.com/cam-mrc-epid/django-q-tree

OR

pip install
svn+https://svn.mrc-epid.cam.ac.uk/private/MRC_Webforms_Python/django-q-tree/
```

Add the following to your INSTALLED_APPS in your django project settings

```
'q_tree',
'polymorphic',
'polymorphic_tree',
'mptt',
```

Run migrations

```
python manage.py migrate
```

To be able to view the XML via the browser you need to add the following url to your project urls.py

```
url(r'^xml_out.xml/', 'q_tree.views.xml_view'),
```

Usage

Admin Interface

You can create Questionnaire, Section, etc. objects via the Admin interface. These can be copy'n pasted and drag and dropped around the interface. At present the copy/paste pastes a single object in place and does not include the children. I don't think it would be a big job to add this functionality. I'll give it a go if I have time.

Importing XML

To import xml

```
    python manage.py importxml <location of xml file> <name>
```

The name is unimportant but must a string. The file location should be the absolute location of your xml file e.g. C:/Data/xmlfiles/myfile.xml without any quotes or spaces. Note the forward slashes.

Exporting XML

To export xml

```
python manage.py exportxml <q_id>
```

where `<q_id>` is the Questionnaire's Q id. You will see this value in the Questionnaire's admin interface. You can set it to what you want. This will output XML to standard output. It can be saved to a file like this

```
python manage.py exportxml <q_id> > SomeFile.xml
```

The XML can be renderered via a browser and saved in that way using the url

```
http://some-server:port/xml_out.xml?q_id=<q_id>
```

where the q_id is the Questionnaire Q id value which can be set in the admin interface.

App creator

The idea of the app creator was the possibility of generating a Django app direct from XML. This is not really possible as TextNodes often provide labels that are not attached to the Question. This function will build models for the XML file's questions and also integrate it into the admin interface. It would be possible to build django forms direct from the XML if labels attached to the questions.

To create an app from an XML file

```
python manage.py buildapp <location of xml file> <appname>
```

This time the name will be the name of the app itself and it will be placed in your project folder. File location as in the importing xml section above.

Project Structure

This a standard Django app which needs to be installed in a project.

The app has the following structure:

q_tree/management/commands This folder contains the management commands for the project.

q_tree/templates/ contains templates which provide the XML output and also 3 templates that override the admin screens default of django-polymorphic-tree so the checkboxes are available and the copy/paste action will work.

q_tree/admin.py contains the admin screen set up. It also contains the copy/paste admin action function.

q_tree/models.py contains the models for the project. They subclass django-polymorphic-tree models. A BaseTreeNode class is the superclass for the Questionnaire, Section, QuestionGroup and Question classes. This common class is used in the admin. Admin change-lists work for a single model and in the this case we use the BaseTreeNode model so that we can list all our object types in a single view. The BaseTreeNode has 2 custom model managers. One provided by django-polymorphic-tree and another from model_utils which allows the copy/paste function to access subclasses from the BaseTreeNode superclass for admin actions.

q_tree/views.py contains a single view that outputs the XML for the given Q id of a questionnaire (see Usage).

Dependencies

django-mptt # A dependency of django-polymorphic-tree.

django-polymorphic # A dependency of django-polymorphic-tree.

django-polymorphic-tree # This provides the allows the tree structure that mptt allows but adds the possibility of using different models within the tree.

beautifulsoup4' # Used in the exportxml command to format the XML.