

# Great Moonstone Oxen of the First and Forsaken Oceans

Taylor McKinney	Matt Stringer	Cesar Cantu
Spencer Reynoso	Ben Bowley-Bryant	David Denton

July 11, 2013

## INTRODUCTION

---

### **What is the problem?**

It is often easy to forget the many crises occurring throughout the world. This site is designed to gather the data on every crisis, and present it in a simple format to keep them in the digitally logged. This site gathers details about victims, along with information about organizations and people involved.

### **What are the use cases?**

There is increasing availability of internet access worldwide and this site makes use of this to share news regarding crises and bring help to victims. This site can be a good source for anyone who wants to be informed and has a desire to help. This site can provide contact information for people and organizations involved. People who want to help can look to our site for the best ways to get involved, whether it be donating to the right charities or volunteering at a local site.

## IMPLEMENTATION

---

### Source Code

Most of the code, including the *importScript* and *export* scripts, are done in Python. The website is delivered using Django and the site is designed using Twitter's Bootstrap. The data is stored to a MySQL server on campus. Most of the separate groups have already shared data, and our database already holds data from all of the shared groups, as well as our own.

### Folder Structure

The model of the data structure can be seen below.

```
wcdb/  
  crises/  
  scripts/  
  static/  
    css/  
    html/  
    img/  
    js/  
    templates/  
    xml/
```

Figure 1. The Folder Layout

The root folder *must* be named *wcdb*. If downloading from Github, rename the root to *wcdb*. Inside the *wcdb* folder, there are three folders *crises*, *scripts*, and *static*.

The crisis folder contains the crisis application and the standard Django files, including *models.py*, which has details about the table structure, and

*tests.py*, which contains most of our unit tests.

The scripts folder contains the scripts *importScript* and *export*. The *importScript* is named to avoid conflicts with the Python keyword. The script *export* will likely be renamed to a similar convention soon.

Inside the static folder, there are the folders *css*, *html*, *img*, *js*, *templates*, and *xml*. The *css* folder contains the CSS files associated with Twitter's Bootstrap.

The *js* folder holds javascript files from Twitter's Bootstrap. This javascript animates our navigation sidebar, highlighting the row when a link is hovered over. The *templates* folder holds just a single HTML file for now that is not in use. Later, this folder will be used to hold templated HTML data that will be used recurringly throughout the site. The *xml* folder currently holds several XML that were temporarily used during development. The only important file is *WorldCrisis.xsd.xml*. This file is the schema our XML data tests against in our *importScript* and *export*.

The *html* folder contains all of our content pages. The file *base.html* is similar to a template. It contains most of the HTML data that can be seen on the site, with several blocks filled in with content from other pages. The other pages, excluding *export.html* and *export.xml* all extend *base.html*. When Django is rendering a page, it will start by rendering data from the page it extends. Since all of our pages extend *base.html*, all of our pages use the *html* found there. When Django is filling in content from *base.html* and encounters a block, it will fill in this block with data from the page it was first passed. The *crisis*, *organization*, and *person* pages are our static pages. They hold information that will be filled into the template by Django. The static pages are still being delivered by Django.

## Data Model

The core of our data model starts with the tables *Crisis*, *Organizations*, *Person*. All three tables have a many to many relation between each other. There is a *Common* table to hold data that all three models share. Crisis, Organizations, and Person can have 0 or 1 Common objects. There is an abstract model, *AbstractListType*. The database will never write an *AbstractListType*, but abstract types are useful when scripting. The types *CommonListType* and *CrisisListType* both inherit from *AbstractListType*. *CrisisListType* will hold data for Crisis objects, and *CommonListType* will hold data for Common.

## Import

Import is implemented as a Python script, in the scripts folder. It is named `importScript.py` to avoid naming conflicts with the Python keyword. It reads in an XML file as input, parse the information, and stores the applicable data into the database for future viewing. The XML file must conform to the schema. Otherwise, no data will be imported. Import is password protected, and only site administrators may run this script. Running import is done on the website. There is a link in the navbar that reads Import. From this page, an administrator may select a file and upload the information. For test use, the password is **downing**, without the quotes. The cookie will expire five minutes after logging in or immediately after closing the browser.

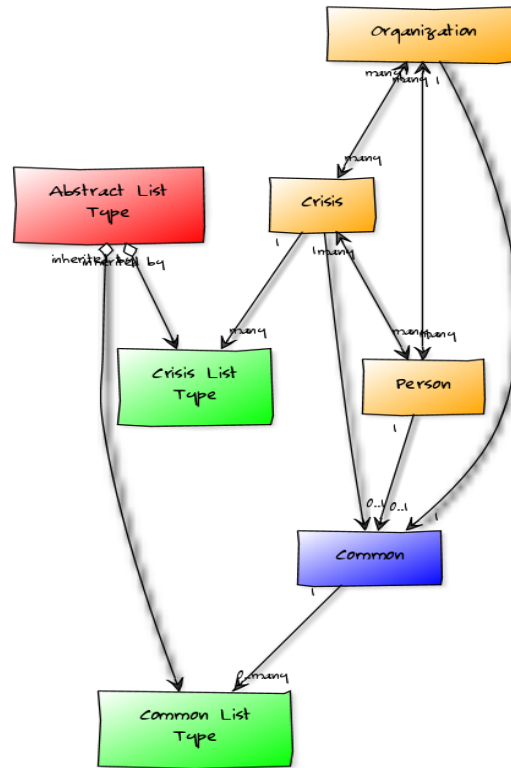


Figure 2. A diagram of our data model

## **Export**

Export is implemented as a Python script in the scripts folder. (It is named `export.py`, although it will be renamed during a later stage to be consistent with `importScript.py`.) Export the data stored in the database and writes it into a new XML file conforming to the schema. As export does not require administrator access, any user can export this database by clicking the Export link from the navigation bar.

## DESIGN

---

### XML Schema

Although our class has been divided into separate groups for this project, our group agreed to share an XML schema between groups. The latest version of the schema can be found at this url.

<https://github.com/aaronj1335/cs373-wcdb1-schema/blob/master/WorldCrises.xsd.xml>

The entire XML file is wrapped in a *WorldCrisis* object. Within the *WorldCrisis* object, there are one or many *Crisis*, *Organization*, and *Person* elements. Each of those elements have uniquely typed key types, named *CrisisKey*, *OrgKey*, and *PersonKey*. There are wrappers for containers of these objects that can contain a list, i.e., a Crises container for multiple Crisis objects, etc. Furthermore, there are type definitions for each type: *CrisisType*, *PersonType*, *OrgType*.

*CommonType* defines data related to Crisis, Person, and Organization, such as links, images, and videos. There is a *ListType* definition that contains a list of XML tokens to facilitate objects with lists of data.

## TESTING

---

Testing is done using tools provided by Django and unittest. To run the tests, run the command **python manage.py test**. This command will run the unit tests using SQLite3 to take advantage of faster unit testing. MySQL should be used for testing in later phases of development to ensure the code is running properly on our production environment. The ideal solution is to create two testing environments: one that runs unit tests on SQLite3 for performance purposes, the second runs tests on MySQL before pushing. This allows us to run quick unit tests while working and ensure the site works on our production environment.

Django extends the standard unittesting framework, providing several hundred tests against the back end. In addition, a number of unit tests are added for the scripts importScript and export. There are plans to implement further testing for each view. As the site only shows static pages, the views can be checked manually.



## FUTURE GOALS

---

Our primary future goal is to get the site presenting data from the database dynamically. Our site is inspired by IMBD and will have similar functionality, including the following:

- Each page should have a gallery for its related images
- A search page, where users can search for any key
- Organize pages by filters, like date and location
- A short list of recent events on the home page based on how kind they are
- Improve the display of data, including charts and graphs

Some other desired features:

- A "Kindness Slider", where users can vote and rate organizations and people
- A proper logo for our team and site
- A random crisis action, to show people random crises
- A sympathy button for crises
- User profiles