CIVET Documentation

Release beta-0.7

Philip A. Schrodt
Parus Analytics LLC
Charlottesville, VA USA
schrodt735@gmail.com

August 16, 2015

CONTENTS

Preface

This is the documentation for the CIVET—Contentious Incident Variable Entry Template—data entry system. CIVET is being developed by the NSF-sponsored project titled "A Method for Leveraging Public Information Sources for Social Science Research" which is creating tools to improve the efficiency of data generation in the social sciences, with an initial focus on coding event data in the domain of contentious politics.

The system is deployed as a Django application; it should be possible to get this working by installing Django on a local machine and copying the directory `djcivet_site`.

We are very interested in feedback on this system, including any bugs you encounter (please let us know what operating system (e.g. Windows, OS-X) and browser (e.g. FireFox, Explorer, Chrome) you were using), aspects of the manual that are unclear (and features that appear too complex), and additional features that would be useful. Please send any suggestions to schrodt735@gmail.com.

Link to the software on GitHub

Acknowledgements

The development of CIVET is funded by the U.S. National Science Foundation Office of Multidisciplinary Activities in the Directorate for Social, Behavioral & Economic Sciences, Award 1338470 and the Odum Institute at the University of North Carolina at Chapel Hill with additional assistance from Parus Analytics.

CHAPTER

ONE

INTRODUCTION

This is the documentation for the beta version of the Civet [1]—Contentious Incident Variable Entry Template—customizable data entry system. Civet is being developed by the NSF-sponsored project titled "A Method for Leveraging Public Information Sources for Social Science Research" which is creating tools to improve the efficiency of data generation in the social sciences. The project has an initial focus on coding event data in the domain of contentious politics, but we expect that these tools will be relevant in a number of data-generation domains.

The core objective of Civet is to provide a reasonably simple—yes, simple—set of commands that will allow a user to set up a web-based coding environment without the need to master the likes of HTML, CSS and Javascript. As currently implemented, the system is a rather ugly prototype; it will also be evolving as we add additional elements. Nonetheless, the system should now be useable for coding.

Civet is implemented in the widely-used and well documented Python-based Django system [2] which is widely available on various cloud platforms: a rather extended list of "Django-friendly" hosting services can be found at

> https://code.djangoproject.com/wiki/DjangoFriendlyWebHosts

The complete Civet code is licensed as open source under the MIT license and provided on GitHub at https://github.com/civet-software .

Civet currently has two modes:

Coding form template: This is a template-based for setting up a web-based coding form which implements several of the common HTML data entry formats and exports the resulting data as a tab-delimited text file. This is fully functional and should be useable for small projects.

Text annotation/extraction: This uses Civet "workspaces" which combine related texts, their metadata, and the coding form. Workspaces allow for manual and automated text annotation, then the ability to extract various types of information into the fields of a coding form.

1.1 Program Navigation Placeholders

Civet is currently under development and not all of the options have been fully implemented. If you see a page with a message of the form

```
The option [something] has yet to be implemented.  Use the back arrow
in your browser to return to the previous screen.
```

you have encountered one of those options: as noted, just use the "Back" option in your browser to return to the previous screen.

[1] http://en.wikipedia.org/wiki/Civet

[2] An earlier prototype was implemented in the `Flask` framework: see Appendix 4

1.2 Documentation

Documentation is maintained using the Sphinx system, which provides both an on-line version and a reasonably-well-formatted PDF version. There are links to both of these compiled versions on the home page; the `.rst` source texts for the documentation are in the directory *djcivet_site/docs*. That directory contains a Sphinx *Makefile* so revisions can be compiled using the standard command `make html latexpdf`.

The on-line documentation currently resides at the site http://civet.parusanalytics.com/civetdocs/; [3] a PDF version can be downloaded by clicking the `Download PDF` link on the home page. [4]

[3] This is a bug, not a feature: there is presumably a way of accessing these at *djcivet_site/docs/_build/html/*, or somewhere else within the *djcivet_site/* directory in a manner that has them correctly rendered, but I haven't figured it out yet. Fixes are welcome.

[4] This is handled in `views.download_pdfdocs()`: it first looks for the PDF version of the documentation in *docs/_build/latex/civetdoc.pdf*, which is where the most current version is likely to be located when the documentation was produced using the `make latexpdf` command in the *docs/* directory. If that isn't present, it checks the */static/* directory: this can be used in deployments in order to avoid uploading *docs/*. If neither is available, it gets the copy posted at http://civet.parusanalytics.com/, which may or may not correspond exactly to the version being used depending on what modifications have been made. The command `make movepdf` will copy *civetdoc.pdf* from *_build/latex* to */static/*

CHAPTER

TWO

INSTALLING CIVET

To date we've only installed the system on Macintosh computers, though the only difference between a Macintosh installation and other installations should be the installation of the Django system.

On Macintoshes running OS-X 9 and 10, the required Python 2.7 comes pre-installed. The `pip` installation program may also be pre-installed—I'm having trouble determining this from the Web, and forget whether I had to install it when I last upgraded—but if not, install that.

1. In the Terminal, run `sudo pip install Django`: you will need administrative access to do this.
2. Download the Civet system from https://github.com/civet-software/CIVET-Django, unzip the folder and put it wherever you would like
3. In the Terminal, change the directory so that you are in the folder *Django_CIVET/djcivet_site*
4. In the Terminal, enter `python manage.py runserver`
5. In a browser, enter the URL http://127.0.0.1:8000/djciv_data/

At this point you should see the Civet home screen

2.1 Modifying the default installation

Because CIVET is still in beta, the version on GitHub is the one being used for development. To deploy the system for active coding, you will probably want to make the following changes:

1. **In the file *djcivet_site/djcivet_site/settings.py*, set `DEBUG = False`.** This will

   It is appropriate to note the Django documentation advice on this:

   ```
   Never deploy a site into production with DEBUG turned on.

   Did you catch that? NEVER deploy a site into production with DEBUG turned on.
   ```

   As the Django documentation discusses in detail, with `DEBUG = True` any errors will generate an error page containing extensive internal detail about your site. With `DEBUG = False`, the user just sees a `Page not found` error.

CIVET

Contentious Incident Variable Entry Template

This is the home page for a prototype of the CIVET—Contentious Incident Variable Entry Template—data entry system. CIVET is being developed by the NSF-sponsored project titled "A Method for Leveraging Public Information Sources for Social Science Research" which is creating tools to improve the efficiency of data generation in the social sciences, with an initial focus on coding event data in the domain of contentious politics.

The system you are looking at is a rather ugly *prototype* and provides a testbed of testing the functionality of some of various elements we expect to have in the final system. It will also be evolving as we add additional elements.

The prototype is currently deployed as a "Django" (`https://www.djangoproject.com/start/overview/`) application; it should be possible to get this working by installing Django on a local machine and copying the directory `djcivet_site`. We currently expect the operational version will be deployed on a UNC server though the code will be available if you wish to create your own local or cloud versions.

Click one of the following links to use the system

Read coding form template only **Read coding form and text collection**

Manage text collections **Set preferences**

Download documentation (PDF)

Acknowledgements

The development of CIVET is funded by the U.S. National Science Foundation Office of Multidisciplinary Activities in the Directorate for Social, Behavioral & Economic Sciences, Award 1338470 and the Odum Institute at the University of North Carolina at Chapel Hill with additional assistance from Parus Analytics.

Last update: 28 July 2015

CHAPTER

THREE

HOME PAGE OPTIONS

The home page has the following links:

Read coding form: Civet reads a coding form template without using a workspace: this is used if you want to use the web coding form without annotated texts. This option can also be used when debugging coding forms. Further instructions for the template-only mode are given in Section [sec:template].

Read workspace: Civet reads a set of text collections and their associated coding form from a zipped file: this mode allows for text annotation and extraction and is described in more detail in Section [sec:workspace], [sec:annotate] and [sec:coding].

Manage workspace: This links to various utilities that operate on workspace files (Section [sec:workspace]) including downloading the coded data as a tab-delimited file, editing the meta-data, and adding comments to the file. [Beta 0.7: only the data download is implemented]

Set preferences: This goes to a page where various program preferences can be set manually. [Beta 0.7: not currently implemented]

Documentation

On-line manual Links to an HTML version of the documentation

Download PDF This downloads a PDF file with the documentation.

3.1 File selection

The first three modes go to a file selection screen.

This provides the following options:

Choose file: Select a file containing a coding form template or workspace, then read this into the system by clicking the `Read file` button.

Coder: Any text entered here—typically a coder name or ID—will be included as metadata with any annotations or cases coded. This field is optional.

Demo file: Read the simple demonstration files built into the system. [1]

Download demonstration file: This downloads a template or workspace demonstration file, which can be used as an example.

[1] These files are named `CIVET.demo.template.txt` and `CIVET.extract.demo.zip` in the directory `djcivet_site/djciv_data/static/djciv_data/` and can be modified there.

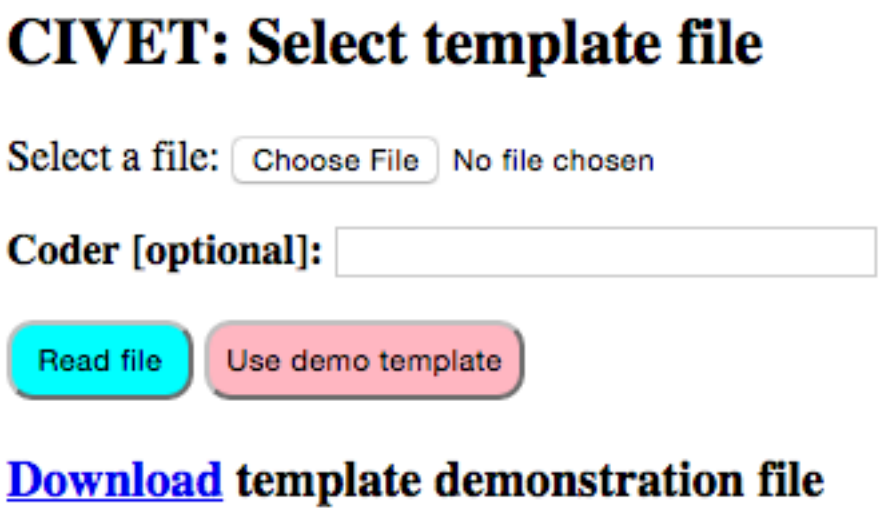


Fig. 3.1: Civet file selection screen

CHAPTER

FOUR

CIVET CODING FORM TEMPLATES

A CIVET template file specifies the individual components of the form: these are the familiar components from web forms but the syntax used to specify them is simpler than what you will find in HTML.

CIVET is simply adding these controls to an HTML `<form>` and, as with all things HTML, most of the placement of the fields is handled by the browser. [1] CIVET provides some limited formatting through the insertion of text and line breaks, and with some experimenting you should be able to keep the form from being too ugly.

The template file should be a simple text file—most systems are happier if this ends in the suffix `.txt`—similar to that used in an *R* or *Stata* script (that is, not a formatted file such as that produced by *MS-Word*). Appendix 1 gives an example of a template file, and the code for this can also be downloaded from a link in the program.

4.1 Simple Template-Based Data Entry Form

The basic data entry form just uses the presumably familiar standard HTML data entry fields and should be self-explanatory.

To save a set of coded fields, click one of the buttons which follow the title `Options after saving:`

Code another case: Save, then return to the same form

Download data: Save, then download data as a tab-delimited text file

The `Download CIVET data` page provides a text box for a file name, and the `Download file` button downloads the coded data. Use the *Start new data file* link to re-start the coding and the *Continue coding with this file* link to continue adding to the existing records.

- The .txt file is tab-delimited and contains the variable names in the first line.
- If the file name does not end in ".txt," this suffix will be added.

4.2 Command formats

Commands generally have the following format

```
command: entry-title [var-name] options
comma-delimited list
```

[1] Writing in HTML5 and CSS, one can actually exercise a very fine degree of control over the placement, but if you are comfortable with that sort of code, you presumably aren't using CIVET in the first place. That said, you can see the HTML generated by CIVET by using the *View source* option in your browser, then save it as a file using *Save Page As...* and that could provide a starting point for creating prettier code.

Download CIVET data:

Enter name of output file:

civet.output.txt [".txt" suffix will automatically be added]

Download file

To continue, use one of the following links:

- Start new data file
- Continue coding
- Return to home page

To quit, just close the window

Commands vary in how many of these components they have, but all follow this general pattern.

Each command ends with a blank line (or, if you prefer, the commands are separated by blank lines.)

Commands can also be cancelled by adding a "-" in front of the command: this will cancel the entire command, that is, all of the lines associated with the command, not just the first line. For visual symmetry, a "+" in front of the command "activates" it, though the command will also be active without the plus.

"#" denotes a comment: anything following a "#" is ignored, so lines beginning with "#" are completely ignored.

4.2.1 Items in template specification

The commands involve one or more of the following items:

entry-title This is the title of data entry field. If this ends with `/` a line-break (`<br>`) is inserted after the text. The titles are escaped: at present the characters <, >and the single and double quotes are replaced with the equivalent HTML entities `&lt;`, `&gt;` `&quot;` and `&rsquo;`. [2] The **entry-title** field cannot contain the characters "[" or "]"—if these are present they will be interpreted as bounding the **var-name** field—but the escaped versions "\[" and "\]" are allowed.

var-name The text of the variable name for this field; this will be used in the first line of the `.csv` output file

comma-delimited-option-list A list of the items that can be selected, separated by commas. A '*' at the beginning of the item means that it will be initially selected.

comma-delimited-var-name-list A list of items which appear in **var-name** fields, separated by commas.

page-text Any text

number An integer

[2] In the current implementation, named HTML entities such as `&copy;` and `&euro;` can be included and should produce the correct character. At present numbered entities such as `&#91;`—the HTML equivalent of ']'—do not work since the # is interpreted as a comment delimiter: depending on whether there is demand for this feature, the system could provide a way around this.

4.2.2 Errors in template commands

There is a fair amount of error trapping as the commands are processed; any problems will reported on a web page. Generally the system will stop after it has encountered the first error rather than reporting all of the errors in the file.

4.3 Templates: Specifying variables

4.3.1 Specifying variables to save

This command gives the variables that will be saved; these can be in any order but each of these must correspond to a `var-name` somewhere in the form, or are one of the special variables discussed below. A tab-delimited version of this list will be the first line of the output file. The command can occur anywhere in the file.

save:
comma-delimited-var-name-list

If the variable name has brackets following it, the *value* of the variable rather than the literal text will be written when the data are written to a tab-delimited file: the value is the string in brackets `[...]` in the annotated coding mode. If there is a variable name inside the brackets, that will be used as the column name for the values; otherwise the regular name will be used: this allows both the literal text and the value to be saved, as in the third example below. If `save` specifies a value output and not is found, a missing value will be used.

Example:

```
save:  worldregion, eyewit, groupname, comments
save:  worldregion [regioncode], eyewit, groupname[], comments
save:  worldregion, eyewit, groupname, groupname [groupcode],
comments
```

4.3.2 constant

Sets the value of a variable to a constant; this can be used in a `save:`

constant: page-text [varname]

Example:

```
constant:  Data set 0.2 [data_id]
```

4.3.3 filename

Sets the default file name for the downloads: this can be changed before downloading. [Beta 0.7: Not yet implemented]

filename: page-text

Example:

```
filename:  our_wonderful_data.csv
```

4.3.4 Special variables

coder : Text entered in the *CIVET template selection* page

date : Current date. this is currently in the form DD-mmm-YYYY but later versions of the system will allow other formats

time : Current time in hh:mm:ss format

4.4 Templates: Data Entry Fields

4.4.1 Checkbox

A simple binary check-box. The value of the variable will be first item in the list when the box is not checked; the second item when the box is checked. The * notation on the second item can be used to specify whether or not the box is initially checked.

select: entry-title [var-name]
comma-delimited-option-list

Example:

```
select:  Eyewitness report?  [eyewit] no,*yes
```

4.4.2 Select from pull-down menu

Pull-down menus—which are called a "select" in HTML—are specified with the syntax

select: entry-title [var-name]
comma-delimited-option-list

Example:

```
select:  Region [worldregion] North America, South America, Europe,
*Africa, Middle East, Asia
```

4.4.3 Radio buttons

A series of radio buttons are specified with the syntax

radio: entry-title [var-name]
comma-delimited-option-list

The entry `/` in the option list causes a line-break (`<br>`) to be inserted

Example:

```
radio:  Region/ [worldregion] North America, South America, Europe,
*Africa, /,Middle East, Asia
```

4.4.4 Enter single line of text

This creates a box for a single line of text (HTML '' type=text''). The `width = number` is optional and specifies the size of the text entry box in characters: the default is `width = 32`

textline: entry-title [var-name] width = number
initial-text

Example:

```
textline:  Name of group [groupname] <enter name>
```

4.4.5 Extract single line from annotated text

This creates a box for a single line of text (HTML ‘‘ type=text‘‘) that will interact with annotated text; in addition information can be manually entered or cut-and-pasted into this box. If this command is used in a form that does not have associated annotated text, it behaves the same as `textline` and the `class` information is ignored.

The `class=class-name` is required and specifies the name of the annotation class that the text-entry box is connected with; a class can be associated with multiple text-entry boxes. There are three standard classes:

- `nament`: named-enties, which are determined by capitalization
- `num`: numbers
- `date`: dates

The `width = number` is optional and specifies the size of the text entry box in characters: the default is `width = 32`

textclass: entry-title [var-name] class=class-name width=number
initial-text

Example:

```
textclass:  Name of city [cityname] class=nament <enter city>
```

4.4.6 Enter multiple lines of text

This corresponds to an HTML “TEXTAREA” object. The `rows = number cols = number` is optional and specifies the size of the text entry box in characters: the default is `rows = 4 cols = 80`

textarea: entry-title [var-name] rows = number cols = number
initial-text

Example:

```
textarea:  Comments [comments] rows = 2 cols = 64 - put any
additional comments here -
```

4.5 Templates: Additional Web Page Formatting

4.5.1 Set page title

Sets the title of the web page: that is, the HTML‘‘<title>...</title>‘‘ section of the header.

title: page-text

Example:

```
title:  CIVET-based coding form
```

4.5.2 Insert text

Adds text to the form: the various options follow the usual HTML formats. In interests of simplicity, text is "escaped" so that special characters are not interpreted as HTML: note that this means that in-line mark-up such as `<i>`, `<b>` and `<tt>` will not work, so if you need this activate and use the `html:` command. Also keep in mind that these commands need to be separated by a blank line.

h1: page-text
h2: page-text
h3: page-text
h4: page-text
p: page-text

Example:

```
h1: Primary data set coding form

p:Please enter data in the fields below, and be really, really careful!
```

The simple command

```
p:
```

is useful for putting some space between form elements.

4.5.3 Insert HTML

[This command may or may not be included in the operational version of the system, as it provides some opportunities for mischief. Stay tuned. It is in the code but currently deactivated; if you are installing your own version of the system, it can be activated by changing a single character in the source code.]

Adds arbitrary HTML code without escaping.

html: page-text

4.5.4 Insert a line break

Adds a new line in the form

newline:

CHAPTER

FIVE

CIVET WORKSPACES

CIVET is part of a projected collection of open-source programs designed to work with very large sets of small text files: in the domain of contentious politics these are usually news articles but the issue of managing very large databases of small texts extends well beyond this application. For example, projects analyzing texts from legislative debates, legislation, campaign web sites and blogs all have much the same character when they are studied at a large scale.

In the CIVET system, files containing sets of individual stories are called "collections": these are typically multiple related news stories—"texts"—from which one or more data records—"cases"—are coded. These are stored in a YAML format [1] which is a structured human-readable text file containing a number of data fields; the details of this are given in Appendix 2.

Sets of text collections are grouped into "workspaces" that also contain an associated coding form and, optionally, other information such as user-specified categories that will be used in automatic annotation. The template file begins with the string "`form.`" and uses the category and template commands described in Section [sec:form]. Workspaces are compressed (`.zip`) directories (folders).

In the current configuration of the system, workspace files are uploaded to the system, annotated and/or coded, then downloaded when a session is completed: no data remain on the machine where the CIVET system is running. In a future version, we expect to have an option for persistent data that could be used on a secure server, as well as options for reading these files from a server.

The workspace will generally *not* be downloaded to the same place it was originally: as a standard HTML security feature, the system does not retain any information about where it obtained a file. Instead, it will be downloaded to wherever your system downloads file: for example on the Macintosh this is a folder named `Downloads`. [2] If you wish to replace the original workspace file, this will need to be done manually or with a script operating locally.

There is some limited error checking as the workspace is processed. If errors are found you will get a screen similar to the figure below listing the errors, which must be corrected before the workspace can be used.

Like error messages in all programs, these are self-explanatory [3] though in general errors will occur either when you are processing a workspace for the first time or if you have manually edited it outside of the CIVET system: once a workspace has been successfully read by CIVET the program should not introduce any errors that would be caught at this point. [4]

The program is sensitive to file names:

[1] https://en.wikipedia.org/wiki/YAML

[2] If you read the workspace from the same directory where it will eventually be downloaded, the behavior presumably depends on the operating system: in the case of the Macintosh both the downloaded file and the decompressed versions get a suffix added. E.g. if the original workspace folder is named `test123` with the compressed version `test123.zip`, the system assigns the downloaded version the name `test123 (1).zip` which decompressed to a folder named `test123 (1)`. We are leaving the task of insuring that the original file is not over-written to the operating system and whatever other utilities you might be using to manage workspaces.

[3] Hahaha…just a little programmer joke…

[4] For example, the error in the variable values string in Figure [fig:workerrors] occurs because of the substring `'whois'='Case1-whois',` which should actually be `'whois':'Case1-whois',` but that '=' could only have been introduced through external editing.

The following errors were found in the workspace file "test123 (4).zip"

No "texts:" segment found in the file "CIVyaml_001.yml"

The following string of variable values in caseid Case-001-1

{'typeincid': 'One-sided Violence','suicide': 'no','whois' = 'Case1-whois','_date_': '2000-01-01','location': 'Case1-location [XXX]','natincid': 'explosives ','injuries': 'Case1-injuries[3]','descrp': 'Case1-description',}

cannot be processed because it contains a formatting error. This case occurs in the file "CIVyaml_002.yml"

No "texts:" segment found in the file "CIVyaml_003.yml"

No 'form.*' file was found in the workspace: This is required for coding.

Please correct these and try again

Click here to return to the selection page

- Any file ending with `.yml` is assumed to be a CIVET -formatted collections file
- There should be one and only file beginning with the string `form.`: this specifies the coding form for the workspace
- Any file beginning with `codes.` is assumed to be a *category vocabulary list*. In the file name, "codes." must be followed by a `category` name then a period; the remainder of a "codes" file name can be anything, though typically it will end in `.txt`.
- Any file ending with `.ini` is assumed to be a configuration file [Beta 0.7: Not yet implemented]

Except for these restrictions, the directory can contain additional files of any kind: these will be preserved when the file is downloaded. A workspace file cannot contain subdirectories.

Additional notes on workspaces:

- So long as the YAML formatting is preserved—which should be fairly straightforward—the system is indifferent as to whether editing is done inside or outside of CIVET .
- If the `form` file is missing or contains errors, the system will display the errors it found, then return to the data selection page.
- If you are manually editing the variable values in the `cases` section, any single quotes (`'`) must be "escaped"; that is, replaced with `\'`. This will be done automatically when cases are generated from inside the program.

5.1 Workspace Management

The `Manage workspace` link on the home page will take you first to a workspace selection page, and then to the page shown below. In Beta 0.7, only the `Export data in tab-delimited format/Use save-variable list in the template` is implemented: this will download any coded cases found in the workspace.

CIVET: Workspace Management

Current workspace: CIVET.workspace_example.zip

Select a task from the options below:

- Export data in tab-delimited format
  - Use save-variable list in the template
  - Select the variables to export
- Edit text meta-data
- Add comments to workspace

5.2 User-specified annotation vocabulary using category

The `category` command is used to set up special categories of words that will be color-coded and can be associated with text-extraction fields. The annotation can either be done automatically or by manually selecting the text and using the `Style` pull-down menu in the annotation editor.

> **category:** category-name [color]
> comma-delimited-phrase/code-list or file-name

The `category-name` must be unique and cannot be one of the standard categories `nament, num` or `date`. The program currently accommodates up to 99 categories. [5]

`color` can be any of the 140 named HTML5 colors, [6] a six-digit hexadecimal RGB color (e.g. `6A5ACD` corresponds to the named color "SlateBlue"; the hex notation provides a presumably sufficient choice of 16,777,216 colors), or a two-digit color from the CIVET palette. [7] The palette, shown below, can be accessed by entering the address

> http://127.0.0.1:8000/djciv_data/make_color_list

while the program is running on a dedicated machine. If `[color]` is empty—that is, `[]`—the system uses a color from the standard list in the listed order.

The program will find capitalized versions of the words in the list—in the example below, both "killed" and "Killed" will match—but not all-capitalized versions: "KILLED" would not be matched. A word or phrase can only be in a a single category: putting one in multiple categories will not cause an error, but only the first category evaluated—generally this will occur in the order the categories were entered—will be marked. Words and phrases within a category are evaluated in the order they are listed—see the example in the chapter on annotation— which can be used to establish precedent when words or phrases overlap. At present the program does not allow partial matches, though a facility for this may be added in the future. [8]

The comma-delimited-phrase/code-list can have codes assigned to each of the phrases: these occur in brackets following the phrase and are added to the text during automated markup. The codes can be any character string. Either the phrase or the code or both can be specified in the output. If some of the phrases in the list have codes and others do not, the blank codes will be assigned a null (or, optionally, missing) string.

The vocabulary list can also be read from a file in the workspace. The file name must begin with `codes.+category-name.`; the remainder of the file name can be anything. [9] This be a text file with one phrase per line and the code in brackets; a line beginning with # is treated as a comment.

Example:

[5] **If you need more, this can be changed by allowing more digits in** the `{:02d}` format in the code
'' UserCategories[newcat].append('termst{:02d}'.format(len(UserCategories)))''
in `CIVET_template.make_category()`

[6] See http://www.w3schools.com/html/html colornames.asp

[7] This palette was assembled in a very ad hoc manner, is not color-blind-friendly, and we would be delighted to substitute something better. The list is set as `CIV_template.CatColorList`

[8] If you want it now, delete the test
"`if endx == idx+len(st):`" in `CIVET_utilities.do_string_markup()`.

[9] The period following the category-name is required!: the file name `codes.weapons_mnsa_list.txt` would not be recognized as a valid `codes.` file. Or rather it would be interpreted as applying to a category `weapons_mnsa_list`, not the category `weapons`.

CIVET Default Category Colors

1: Magenta	2: SeaGreen	3: Orchid	4: Brown
5: Purple	6: Gold	7: Olive	8: Slateblue
9: Cyan	10: Thistle	11: CornflowerBlue	12: DarkGray
13: Lime	14: Turquoise	15: SlateGray	16: Tan

Colors shown as text

Plain text Named entity Number Date Magenta SeaGreen Orchid Brown Purple Gold Olive Slateblue Cyan Thistle CornflowerBlue DarkGray Lime Turquoise SlateGray Tan

Close the window to exit

```
category:action [red] killed [1], wounded [2], shot and killed [1],
bombed [3], clashed [3]

category:people [Brown] civilians, workers, authorities, troops,
soldiers, rebels, people, group

category:nationstate [Gold] codes.nationstate.txt

category:weapons [Olive] codes.weapons.mnsa.weaponslist_150724.txt
```

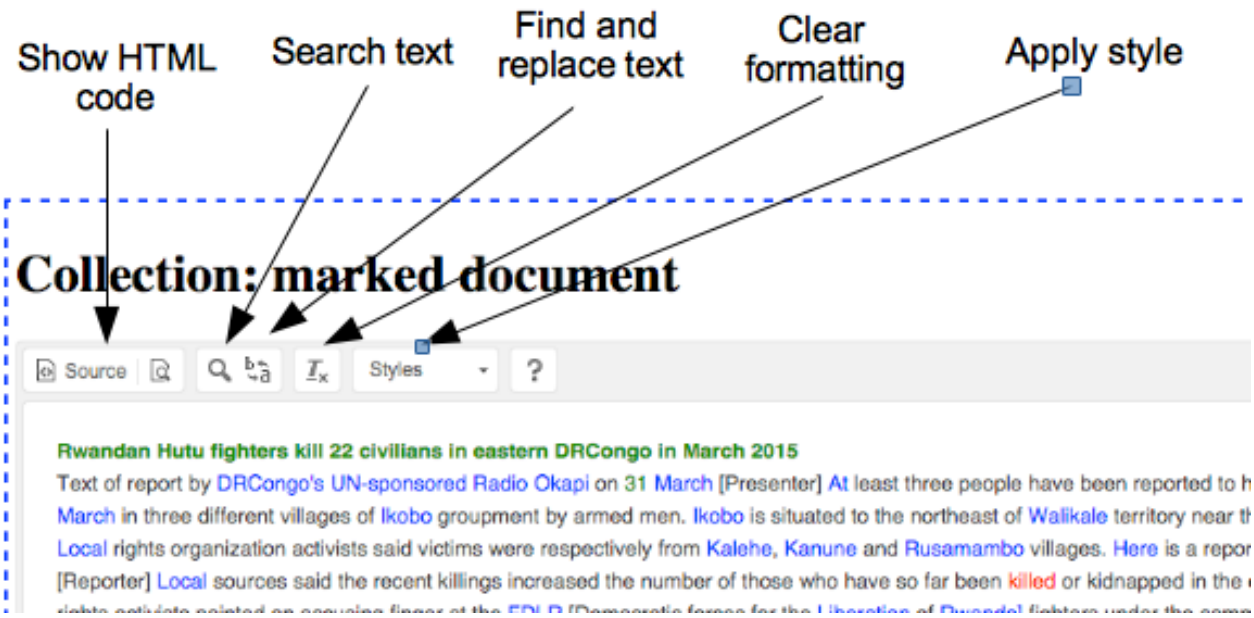
CHAPTER

SIX

ANNOTATION AND EDITING COLLECTIONS

The annotation and editing page for workspace collections implements a minimal version [1] of the Javascript `ckeditor` which allows the texts to be edited and annotated. Editing works as you would expect, including cut/copy/paste options.

Annotation is handled with the `Styles` drop-down menu in the window toolbar which should show both the standard CIVET categories—named-entity, number and date— and any user-specified categories. To annotate, just select the text you want to annotate and then select the annotation to apply.

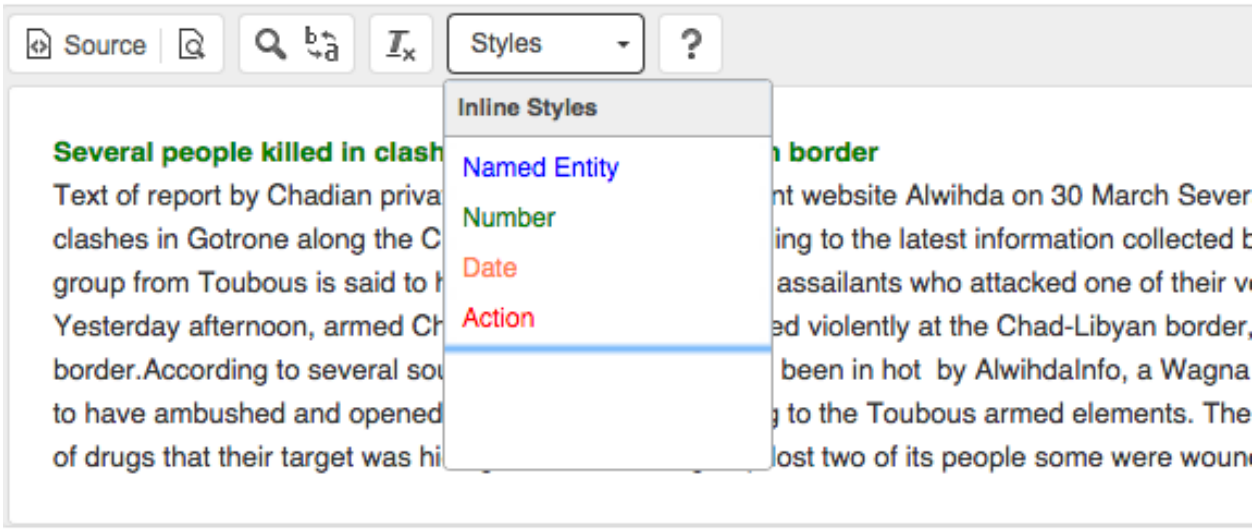


The following options are available on this screen

Annotate the collection:

> This applies the automated markup system which currently annotates the following categories of words and phrases:
>
> **Named-entities:** This is based on capitalization; consecutive capitalized words are combined.
>
> **Numbers:** Digits and numerical words and phrases such as “one” and “two-hundred.”
>
> **User-specified categories:** See the discussion of *categories*

[1] that is, the version of `ckeditor` deliberately uses only a very small set of the features that are available for the editor: if you want to customize this, additional features can easily be added.



Save edits and select new collection:

> This saves whatever annotation has been done to the internal database [2] and returns to the collection selection screen : this option would be used if you are only annotating text rather than coding them. Annotations are saved in the `textmkup` field of the YAML file along with the date of the annotating and the coder ID.

Save edits and code the collection:

> This saves whatever annotation has been done to the internal database and goes to the coding and text extraction page

Discard edits and select new collection:

> This discards the edits and returns to the collection selection screen.

Download workspace and return to home screen:

> This downloads the current workspace without doing any coding.

6.1 Comments on annotation and editing

1. Associated codes in brackets following a term can be edited: when writing variable values, the system will simply be looking for a value in a bracket.
2. A word or phrase can be annotated only once. [3] The user-specified `category` words are annotated before the general named-entity, so if a named entity occurs in a `category`, that will take precedence. Similarly, any numbers that occur in a `category` phrase will be part of the phrase, not separately marked as numbers.

[2] That is, the data is saved on the machine where CIVET is running; it is not saved on your local machine until the workspace is downloaded.
[3] It would be possible to modify the system to allow for phrases to be in multiple categories, but at present this seems like a low priority; such a feature may or may not be included in future versions.

3. Words and phrases in `category` lists are evaluated in the order they are listed, which can be used to establish precedence.

   Consider the sentence

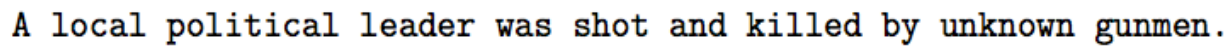


Fig. 6.1: The category listing:

```
category:action [red]
shot and killed [4], killed [1], wounded [2], bombed [3]
```

would result in the annotation

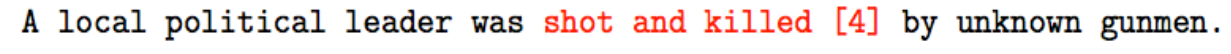


Fig. 6.2: whereas category listing:

```
category:action [red]
killed [1], shot and killed [4], wounded [2], bombed [3]]
```

would result in the annotation

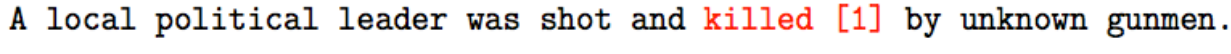


Fig. 6.3: because the "killed" part of the phrase "shot and killed" has already been annotated, and the remainder does not fit any of the patterns.

4. CIVET does not identify a capitalized word as a named-entity if it occurs as a single word and is in the list of common "stop words" in the file

   `djcivet_site/djciv_data/static/djciv_data/CIVET.stopwords.txt`

   In other words, "The" will be included as part of a named-entity in the phrase "The New York Times" but not in the phrase "The village was…"

5. Words referring to numbers such as "one", "ten" and "fifty" have the corresponding numerical value added in brackets following the number; these phrase and their associated values are obtained from the file

   `djcivet_site/djciv_data/static/djciv_data/CIVET.numberwords.txt` [4]

   This file only contains the most commonly-encountered phrases; bracketed values can be added manually as well.

5. At present, CIVET does not recognize leading punctuation—typically quotes—and will not automatically mark named entities or numbers beginning with this: this is on the list of changes for the future. It does handle most trailing punctuation. In named entities, the lower-case prefixes "al-", "bin-" and "ibn-" are recognized as part of a name. [5]

[4] Looking for a little programming exercise?: This needs more development in at least three ways. First, generate all of the standard English equivalents, e.g. "eighty-five", since these follow a simple set of rules. Second, and perhaps more important, allow the user to specify the values for common approximations such as "several,", "many" and "dozens." The second can be done by just editing the file `CIVET.numberwords.txt`, though generally we don't want the user to have to figure out how to do that. Finally, there should probably be some error checking to make sure the value in brackets is actually a number: CIVET will just copy the value in brackets without trying to convert it, but non-numbers will presumably create issues further down the processing pipeline.

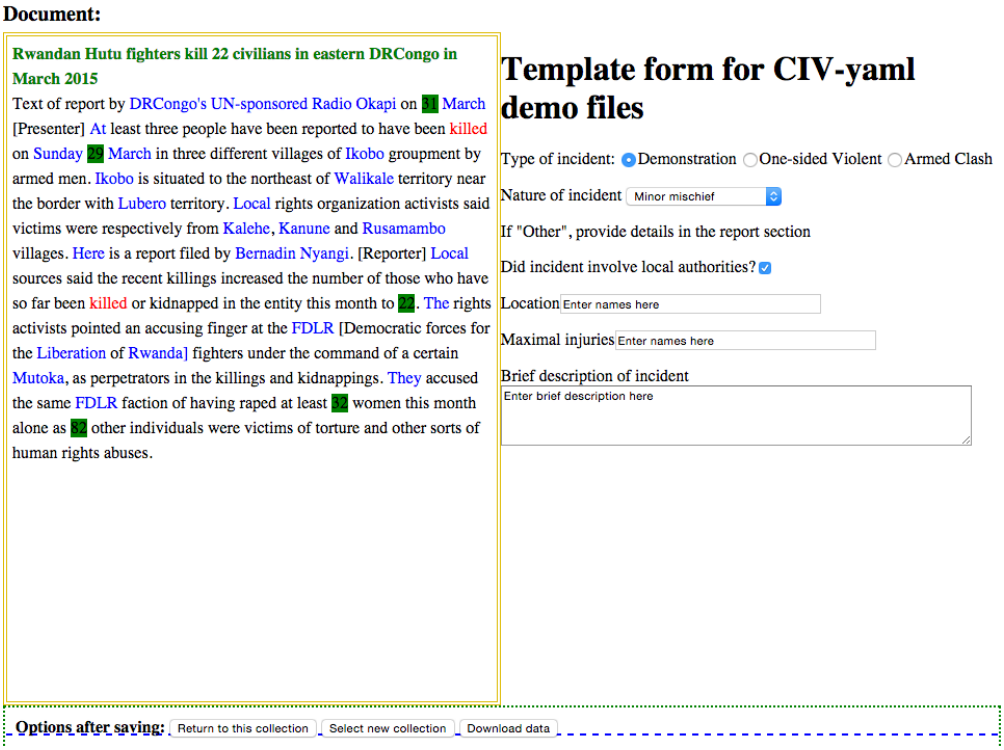
[5] This list can be extended in the regular expression `pat1` in `civet_utilities.do_NE_markup()`.

CHAPTER

SEVEN

CODING AND TEXT EXTRACTION

The CIVET coding form screen in the demonstration version is shown below. [1]



The general operation of the coder/extractor is described below:

1. Clicking a text entry boxes associated with an annotation category will highlight the relevant words in text: In the demonstration version these are

   Location: named-entities

   Maximal injuries: actions

   The 'tab' key cycles between the coding fields, or an option can be selected using the mouse.

2. When an annotated category field is active, all of the words and phrases in the text for that category are changed to red, with the first word highlighted using a green background. The arrow keys can be used to move the highlighted text into the field. These operate as follows:

[1] The form displayed is specified in the file `djcivet_site/djciv_data/static/djciv_data/CIVET.demo.coder.template.txt` and can be modified if you want to experiment.

Right arrow: Highlight the next text in the category [2]

Left arrow: Highlight the previous text in the category

Down arrow: *Replace* the contents of the field with either the currently selected text—this is effectively a single-key shortcut for a copy-and-paste—or, if no text is selected, the highlighted text. [3]

Up arrow: *Append* the contents of the field with either the currently selected text or, if no text is selected, the highlighted text

3. Copy-and-paste from the text to the data fields work as you would expect; text can also be entered manually.
4. To save a set of coded fields, click one of the buttons along the bottom. At present, all three buttons save; later versions add "cancel" and "reset" options. The options are:

   Continue coding this collection: Save the data internally, then return to the same text to code additional cases.

   Code next collection: Save the data internally, then select the next collection in the workspace and go to the annotation screen. [4]

   Select new collection: Save the data internally, then select a new collection

   Download workspace and return to home screen: This downloads the workspace with the coded cases to the local machine. The *Manage workspace* facility can then be used to download any coded cases.

[2] Occasionally you will need to hit the key twice when changing directions: this is a bug, not a feature, and may be corrected at some point. Usually it works the first time. If you would like to try to fix this, look at the Javascript in the file `civet_coder.html`

[3] If you are tabbing between fields and extracting the first highlighted text, you will need to hit down arrow twice: also a bug rather than a feature.

[4] Beta 0.7: In the final version of the program, there will be an option for going to either the annotation or coding screen; the annotation screen will also have a "Next" button.

CHAPTER

EIGHT

PROJECTED FEATURES

CIVET is part of a projected system designed for managing tens-of-thousands, or even millions, of small text files. The transition in the past three decades from paper-based to electronic sources has dramatically increased the amount of information that can potentially be coded, but results in a "drinking from a fire hose" problem where a huge number of false positives must be managed because typically only a very small percentage of the texts obtained for a project actually contain unique codeable events: yields of 1% to 3% are not uncommon. There is very little existing software designed to deal with this situation, since the texts are too large to be treated as nominal variables in a statistical package and too numerous to be treated as documents in a word processor. Consequently large projects typically write customized systems in a language such as perl or Python, but these require programming skills which are not always easily available in the social science community.

We are planning to extend the CIVET workspace format to become the basis of an integrated series of well-documented and user-friendly utilities for dealing with this situation. All of the software will be open-source under the MIT license, and made available to the community on GitHub. These utilities will provide at least the following capabilities:

- Near-duplicate detection which will collect articles which appear to be dealing with the same incident
- Extraction programs for converting common formats such as Lexis-Nexis, Factiva and GigaWord to the CIVET document format.
- Filtering and classification of texts based on one or more of the following methods

  Pattern-based: These will include regular expressions and boolean phrases with proximity measures

  Semi-supervised learning: The system will construct one or more machine-learning models (for example support vector machines) to determine whether an article is relevant based on a set of positive and negative examples provided by the user

  Action-based: These will use either the open source TABARI or PETRARCH political event coders to determine the type of activity being described

  Actor-based: These will use a set of standard lists maintained on a common server of political actors such as nation-states, international organizations and militarized non-state actors

  Geographical: These will use systems such as the open-source Mordecai location resolution system developed by Caerus Analytics.
- Workflow management software for allocating and tracking the coding of incidents in large coding teams; these will use web-based tools so that coders can work from any location and across institutions. We will also provide scripts for interfacing to mySQL installations, GitHub and Dataverse as remote servers.
- Extension of CIVET to allow the various classification tools (actions, actors, and location) to automatically be used in coding forms.
- Semi-automatic conversion of the resulting coded data to the Dataverse format, and more generally integrate the CIVET tools with the Dataverse metadata, APIs and other tools as well as providing an access and authorization protocol modeled on the categories used in Dataverse.

- Development of training materials, both text and video, for the system
- We are currently developing this based on a fixed frame that is 960 pixels wide and 700 pixels high: this will fit easily on a 1024x768 screen. For contemporary equipment this is probably quite conservative but it is not unheard of for data labs to have older equipment so we're going with this at the moment

CHAPTER

NINE

APPENDIX 1: SAMPLE TEMPLATE FILE

```
# CIVET template demonstration file

h1:Ministry of Magic Hogwarts Incident Report

radio: House where incident occurred: [house]
Gryffindor, Hufflepuff, Ravenclaw, *Slytherin

p:

select:Nature of incident [natincid]
*Minor mischief, Unauthorized absence, Accident, Major infraction, Unforgivable Curses, Other

p:If "Other", provide details in the report section

checkbox: Was incident reported to school authorities? [authreport]
No,*Yes

checkbox: Did incident involve muggles? [muggles]
No,Yes

p:

textline: Name of student(s) [names] width=80
Enter names here

p:

textarea:Brief description of incident [descrp] cols = 80
Enter brief description here

p:

textline:Reporting official [reporter] width=40
Enter your name here

h3:Thank you for your assistance; we will contact you by owl should we require more information

save:
_date_, house, natincid, authreport, muggles, names, descrp, reporter
```

This produces the form

Ministry of Magic Hogwarts Incident Report

House where incident occurred: ○ Gryffindor ○ Hufflepuff ○ Ravenclaw ◉ Slytherin

Nature of incident Minor mischief

If "Other", provide details in the report section

Was incident reported to school authorities? ☑ Did incident involve muggles? ☐

Name of student(s) Enter names here

Brief description of incident

Enter brief description here

Reporting official Enter your name here

Thank you for your assistance; we will contact you by owl should we require any additional information

Options after saving: Code another case | Download data

CHAPTER

TEN

APPENDIX 2: INPUT FORMAT

Fields marked with ** are required.

10.1 Collection fields

collid Collection ID, which needs to be unique within the workspace. If this is not provided in the file, collfilename is assigned by the program

collfilename directory and name of the YAML file (without the suffix) where the file was read from; this is assigned by the program

colldate collection date YYYY-MM-DD

****colledit** datetime of editing of this collection [provided by system]

collcmt collection comments

texts one or more related texts

cases zero or more coded records

10.2 Text fields

****textid** unique text ID for CIVET. This needs to be unique within the workspace, and given how collections might get mixed across workspace folders, ideally should be unique for the entire project. If a value for the `text` field is not provided it will be assigned by the program.

****textdate** text date YYYY-MM-DD

textpublisher publisher [any string]

textpubid publisher ID [any string]

textbiblio bibliographic citation

textgeogloc geographical locations

textauthorr author [any string]

textlang language

textlicense copyright notification or other license information

****textlede** lede/headline/abstract—this is a short summary of the article which will be highlighted and also will appear in the sorting routine.

textcmt comment

****textoriginal** original text of the story; this will not be modified by the system

textmkup marked up text: this is the annotated version of the story with any mark-up that has been added either automatically on manually

textmkupdate datetime time of editing of this block [provided by system]

textmkupcoder coder ID

10.3 Case fields

**** caseid** Internal case/event ID. This is assigned by the program and probably should not be changed; external IDs can be entered as variables.

**** casedate** Date and time this case was coded [provided by system]

casecmt comment for case

casecoder coder ID

casevalues This is a string formatted as a Python dictionary which contains pairs of variable names and values

10.4 Date formats

[This has not been consistently implemented in Beta-0.7]

Dates are ISO-8601 (http://en.wikipedia.org/wiki/ISO_8601; http://www.w3.org/TR/NOTE-datetime; https://xkcd.com/1179/; http://www.cl.cam.ac.uk/mgk25/iso-time.html) so generally either

- YYYY-MM-DD
- YYYY-MM-DDThh:mm:ss
- YYYY-MM-DDThh:mm:ss[+-]hh:mm

10.5 Sample File

The following figure shows an example of a simple YAML file; This is a screen capture of a file being edited with *BBEdit*, hence the color mark-up. A workspace demonstration file with several collections can also be downloaded in the program.

```
collid: TestTexts_001
colldate: 2015-06-08
colledit: 2015-06-08
collcmt: Text file source: Apr15.OTH.stories.txt

texts:

  - textid: TestTexts_002_001
    textdate: 2000-01-01
    textpublisher: BBC Monitoring Africa
    textpubid: BBCAP
    textbiblio:  BBCAP00020150401eb4100105
    textgeogloc:
    textlang: English
    textlicense: (c) 2015 The British Broadcasting Corporation. All Rights Reserved. No material may be reproduced except
    with the express permission of The British Broadcasting Corporation.
    textlede: Several people killed in clashes along Chadian-Libyan border
    textcmt:
    textoriginal: |
        Text of report by Chadian privately-owned, pro-government website Alwihda on 30 March

        Several people have been killed and others injured in violent clashes in Gotrone along the Chad-Libyan border. According
         to the latest information collected by AlwihdaInfo, since yesterday [29 March], a heavily armed group from Toubous is
         said to have been in hot pursuit of assailants who attacked one of their vehicles, leading to the death of two of
         their people.

    textmkup: |
        <div class="textblock" data-textid=" TestTexts_002_001"><div class="textlede" style="color:green; font-weight: bold;">
        Several people killed in clashes along Chadian-Libyan border</div><div class="textcontent">Text of report by
        <span style="class:nament; color:blue">Chadian</span> privately-owned, pro-government website
        <span style="class:nament; color:blue">Alwihda</span> on <span style="class:num; color:green">30</span>
        <span style="class:nament; color:blue">March Several</span> people have been
        <span style="class:termst; color:red" title="whacked">killed</span> and others
        <span style="class:termst; color:red" title="whacked">injured</span> in violent clashes in
        <span style="class:nament; color:blue">Gotrone</span> along the
        <span style="class:nament; color:blue">Chad-Libyan</span> border. <span style="class:nament; color:blue">According</span>
        to the latest information collected by <span style="class:nament; color:blue">AlwihdaInfo</span>,
        since yesterday [29 <span style="class:nament; color:blue">March]</span>, a heavily armed group from
        <span style="class:nament; color:blue">Toubous</span> is said to have been in hot pursuit of assailants who
        <span style="class:termst; color:red" title="whacked">attacked</span> one of their vehicles, leading to the death of
        two of their people.
    textmkupdate: 2015-06-08
```

CHAPTER

ELEVEN

APPENDIX 3: SUPPORTING FILES

11.1 Files in `/static/djciv_data`

11.1.1 Files that can be modified using a text editor

CIVET.demo.template.txt: Demonstration template file for simple coding

CIVET.workspace.demo.zip: Demonstration workspace with sample collections, coding form and user-specified coding categories

CIVET.stopwords.txt: Stop words for automatic named-entity annotation

CIVET.numberwords.txt: Number words and phrases for automatic number annotation

civetstyle.css: Style sheet for some of the program (this is modified with the user-specified categories)

11.1.2 Modify at your own risk

ckeditor: This is a `ckeditor` file downloaded from http://ckeditor.com/: if you would like additional features you should be able to create your own and swap it in here.

11.1.3 CIVET Logo

civet_logo.png: Don't like our little guy, or want to put your own mascot here?—this is the place to make the change

11.2 Documentation

CIVET's documentation is maintained using the Sphinx http://sphinx-doc.org/ system. The files are found in the `docs` directory at the outer-most level of the system. The commands:

```
make html
make latexpdf
```

are used to generate the on-line and PDF documentation; the files are found in the `\_build/html` and `\_build/latex` directories.

CHAPTER

TWELVE

APPENDIX 4: PROTOTYPE ON GOOGLE APPLICATION ENGINE

An earlier demonstration version of the program, written in the Flask framework, is deployed as an application on the Google App Engine at http://ace-element-88313.appspot.com/. The "Coding Form Template" option in this program works as described *here*. The code for this version can be downloaded from https://github.com/philip-schrodt/CIVET-Flask

The other option in the program is the "Text-Extraction Demonstration Form" which was a prototype of the full annotation/extraction system. To activate the demo, from the home page, click the link in the line *See a demo of the text-highlighting system by clicking here*

1. Select a text file to edit: you can use either the pull-down menu or radio boxes, then click the `Edit the file button`.
2. Click one of the text entry boxes will highlight the relevant words in text: For demonstration purposes these are words beginning with the letters 'a', 'c', 'd', 'e' and 's'. The 'tab' key cycles between these options, or an option can be selected using the mouse.
3. When a text entry box is active, the first relevant word in the text is highlighted. The right-arrow key will cycle the highlighted word. To copy a highlighted word into the text box, use the down-arrow key.
4. Text can also be selected using the mouse: To copy the selected text into the text box, use the left-arrow key.
5. Cut-and-paste from the text to the date fields work as you would expect
6. Text can also be entered manually.
7. To save a set of coded fields, click one of the buttons along the bottom.

   Return to this case: Save, then return to the same text

   Select new case: Save, then return to the same text

   Download data: Save, then download data as a text file
8. The "CIVET Download" page provides a text box for a file name, and the `Download file` button downloads the coded data. Use the *Start new data file* link to re-start the coding and the *Continue coding with this file* link to continue adding to the existing records.
   - The .txt file contains the variable names in the first line.
   - If the file name does not end in ".txt", this will be added.
9. To quit the program, just close the window: This is a HTML/Javascript security feature which prevents rogue websites from closing windows unless they have created the window.

CHAPTER

THIRTEEN

APPENDIX 5: INSTALLING IN AWS-EB AND DOCKER

13.1 Amazon Web Services Elastic Beanstalk

"Cloud" servers are an increasingly popular low-cost alternative to locally-administered servers. A large number of these options are available; [1] I'm focusing on the Amazon Web Services "Elastic Beanstalk" (AWS-EB) for the simple reason that at the time of this writing AWS provides a generous free trial option, and their instructions worked the first time I tried. [2]

A coherent set of instructions can be found at http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html. Then follow these steps:

1. Get an AWS account: https://aws.amazon.com/

2. In the instructions, skip the steps prior to the heading **Configure your Django application for AWS Elastic Beanstalk** unless you also want to try out the system locally (which is probably a good idea)

3. Download the CIVET system from GitHub: https://github.com/civet-software/CIVET-Django [3]

4. Create a directory that you will use to deploy the system: for consistency with the remaining instructions it should be called *AWS-CIVET* though once you are comfortable with these instructions it could be named something different.

5. In that directory, copy the directory *djcivet_site*. Following the instructions, create directories named *.elasticbeanstalk* and *.ebextensions* [4] and create a file named *requirements.txt* Just copy the contents from the section below; you don't need the `pip freeze` step. Copy the code in the **AWS-EB Configuration Files** section below into the various files

Your directory will now look like

```
AWS-CIVET
|-- .ebextensions
|   `-- 01-django_eb.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- djcivet_site
|   |-- db.sqlite3
|   |-- djcivet_data
|   |-- docs
|   |-- djcivet_site
|   |   |-- __init__.py
```

[1] In particular, Heroku (https://www.heroku.com/) appears to be another Django-friendly option, and also offers free accounts. Using Heroku requires a [free] GitHub account.

[2] Which, ahem, cannot be said for my multiple attempts to get the system running on the comparable Google service, though I'm sure it is possible to do this and would be happy to add instructions once someone has figured it out.

[3] At some point I'll put a "turn-key" directory on GitHub that will have all of the appropriate files. But not yet.

[4] The '.' in front of the file name means these will probably be invisible in most standard views of the *AWS-CIVET* directory: this is a Unix feature, not a bug.

```
    |   |   |-- settings.py
    |   |   |-- urls.py
    |   |   |-- wsgi.py
    |   `-- manage.py
    `-- requirements.txt
```

6. Install the "eb" command-line tool per the instructions found at http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html. Initializing this will require AWS access credentials, a process described at http://docs.aws.amazon.com/general/latest/gr/getting-aws-sec-creds.html.

7. Follow the instructions in the **Deploy your site using AWS Elastic Beanstalk** section to use "eb" in a terminal application. It will take a minute or so for the `eb create` process to complete—there's plenty of feedback—and there is an additional lag before the URL will be recognized.

8. You should now see the CIVET home page at the URL http://aws-civet-dev.elasticbeanstalk.com/djciv_data/ Be sure to include the final "/djciv_data/" or you will get an error. Run through the options with the demonstration files to make sure the site is working. If the site doesn't come up, try reloading a couple of times.

9. When you are finished, enter `eb terminate` and respond to the confirmation prompt with "AWS-CIVET-dev" in order to stop the program.

13.1.1 AWS-EB Configuration Files

In

```
Django==1.8.3
```

In *AWS-CIVET/.elasticbeanstalk/* create the file *config.yml*

```
branch-defaults:
  default:
    environment: AWS-CIVET-dev
global:
  application_name: AWS_CIVET
  default_ec2_keyname: aws-civet
  default_platform: Python 2.7
  default_region: us-east-1
  profile: eb-cli
  sc: null
```

In *AWS_CIVET/.ebextensions/* create the file *01-djcivet_site.config*

```
container_commands:
    01_collectstatic:
      command: "source /opt/python/run/venv/bin/activate && python djcivet_site/manage.py collectstat

option_settings:
  "aws:elasticbeanstalk:application:environment":
    DJANGO_SETTINGS_MODULE: "djcivet_site.settings"
    PYTHONPATH: "/opt/python/current/app/djcivet_site:$PYTHONPATH"
  "aws:elasticbeanstalk:container:python":
    WSGIPath: "djcivet_site/djcivet_site/wsgi.py"
```

In *djcivet_site/djcivet_site/settings.py*

- Set `DEBUG = False`
- Change SECRET_KEY since the downloaded version isn't exactly secret

13.1.2 Handling of *static* files

As even a brief perusal of the web will affirm, the handling of *static* files in production versions of Django is, well, complicated. After joining legions of programmers past, present and future in beating my head against the wall on trying to get CIVET to access files internally in production as it does in the development mode, I gave up [5] and put the static resources referenced from inside templates on a directory on an external server, specifically *http://civet.parusanalytics.com/civet_static/* Files that are read in *views.py* remain in the *static/djciv_data* folder, which works in both the development and production modes.

If you would like to modify the static files in the system—the main target would be *CKEditor*, unless you find our mascot too insufferably cute—you can move this material (the contents of the directory *static/djciv_data* in the distribution) to the server of your choice: just change the address in `settings.STATIC_SOURCE` to point to the new location. [6]

13.2 Docker

Docker (https://www.docker.com/) is a highly popular, rapidly evolving [7] "containerization" system which will ultimately simplify the secure deployment of software in a wide variety of different systems. Briefly, "containers" are a more efficient extension of the concept of virtual machines —computers running programs which simulate the operation of other computers—by packaging all of the required software in an "image" file that is able to run on any system capable of running Docker. Because the operations within a container can be isolated from the host machine, and the contents of the container can be inspected and verified, this should provide a more secure (and efficient) environment than situations where a variety of software needs to be installed in order for a system to run, and that software potentially has access to all of the resources of the system. [9] Hence the excitement.

To date, I have successfully gotten the Docker container described below to run CIVET in development mode as a container on my Macintosh; I attempted to get it running on the Google Cloud but was unsuccessful; I have not tried any other configurations. As always, I will be happy to incorporate any additional suggestions into this documentation.

The guide I used for the deployment is http://michal.karzynski.pl/blog/2015/04/19/packaging-django-applications-as-docker-container-images/. This was not the first one I tried, and as indicated above, Docker is still evolving so you should make certain you are using a recent set of guides (and the instructions here may break sooner rather than later.)

Using Karzynski as a guide, here are the steps:

1. If you aren't already using Docker, get a Docker account—there is a free option—and install Docker: the instructions for this will vary depending on your operating system; Karzynski's instructions are just for Linux.

2. Set-up a directory to hold the Docker project—I called this *Docker-CIVET*, which corresponds to Karzynski's local directory *dockyard*. I'll be using Karzynski's Docker image name DOCKYARD.

3. Copy the directory *djcivet_site* into *Docker-CIVET*.

4. In *Docker-CIVET*, create the *docker-entrypoint.sh* and *Dockerfile* files from the code given below. Your directory will now look like

```
Docker-CIVET
|-- docker-entrypoint.sh
|-- Dockerfile
```

[5] Or simply took the approach that the Django system clearly prefers, depending on your perspective

[6] An apparently popular approach for handling this is to use an AWS S3 server instance for external storage of static files: there are multiple descriptions on the Web describing how to do this. As it involves quite a few steps and I've got a perfectly good server already set up in the cloud, I went with that route instead.

[7] Which is to say, a whole lot of moving parts which don't quite always play well together and inconsistently documented: see http://blog.circleci.com/its-the-future/, http://blog.circleci.com/it-really-is-the-future/, and https://valdhaus.co/writings/docker-misconceptions/ [8]

[9] Or as the situation was recently explicated at our local software development meet-up, in reference to a certain institution that does not have a campus but "Grounds", and I am not referring to Starbucks, "So which is it with your sysadmins? They want to make sure Docker is deployed securely? Well, there are plenty of ways to do that. Or they just don't want to do any work? Then you've got a different set of problems."

```
    |-- djcivet_site
    |   |-- db.sqlite3
    |   |-- djcivet_data
    |   |-- docs
    |   |-- djcivet_site
    |   |   |-- __init__.py
    |   |   |-- settings.py
    |   |   |-- urls.py
    |   |   |-- wsgi.py
    |   `-- manage.py
    `-- requirements.txt
```

5. Follow the remaining instructions to build and test the container with the `user-name` from your Docker account and the `image-name` of your choice (e.g. `docker-civet`).

13.2.1 Contents of *docker-entrypoint.sh*

```
#!/bin/bash
python manage.py migrate                  # Apply database migrations
python manage.py collectstatic --noinput  # Collect static files

# Prepare log files and start outputting logs to stdout
touch /srv/logs/gunicorn.log
touch /srv/logs/access.log
tail -n 0 -f /srv/logs/*.log &

# Start Gunicorn processes
echo Starting Gunicorn.
exec gunicorn djcivet_site.wsgi:application \
    --name djcivet_site \
    --bind 0.0.0.0:8000 \
    --workers 3 \
    --log-level=info \
    --log-file=/srv/logs/gunicorn.log \
    --access-logfile=/srv/logs/access.log \
    "$@"
```

13.2.2 Contents of *Dockerfile*

```
###############################################################
# Dockerfile to run a Django-based web application
# Based on an Ubuntu Image
###############################################################

# Set the base image to use to Ubuntu
FROM ubuntu:14.04

# Set the file maintainer (your name - the file's author)
MAINTAINER Parus Analytics

# Set env variables used in this Dockerfile (add a unique prefix, such as DOCKYARD)
# Local directory with project source
ENV DOCKYARD_SRC=djcivet_site
# Directory in container for all project files
ENV DOCKYARD_SRVHOME=/srv
```

```
# Directory in container for project source files
ENV DOCKYARD_SRVPROJ=/srv/djcivet_site

# Update the default application repository sources list
RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y python python-pip

# Create application subdirectories
WORKDIR $DOCKYARD_SRVHOME
RUN mkdir media static logs
VOLUME ["$DOCKYARD_SRVHOME/media/", "$DOCKYARD_SRVHOME/logs/"]

# Copy application source code to SRCDIR
COPY $DOCKYARD_SRC $DOCKYARD_SRVPROJ

# Install Python dependencies
#RUN pip install -r $DOCKYARD_SRVPROJ/requirements.txt
RUN pip install Django
RUN pip install gunicorn
# Port to expose
EXPOSE 8000

# Copy entrypoint script into the image
WORKDIR $DOCKYARD_SRVPROJ
COPY ./docker-entrypoint.sh /
ENTRYPOINT ["/docker-entrypoint.sh"]
```