



# **CIVET Documentation**

*Release beta-0.9.1*

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

August 17, 2016



## CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Program Navigation Placeholders . . . . .	3
1.2	Status of the Program: 31 August 2015 . . . . .	4
1.3	Status of the Program: August 2016 . . . . .	4
1.4	Documentation . . . . .	5
<b>2</b>	<b>Installing CIVET</b>	<b>7</b>
2.1	Modifying the default installation . . . . .	7
<b>3</b>	<b>Authentication</b>	<b>9</b>
3.1	Creating a superuser . . . . .	9
3.2	Additional notes . . . . .	9
<b>4</b>	<b>Home Page Options</b>	<b>11</b>
4.1	File selection . . . . .	11
<b>5</b>	<b>CIVET Coding Form Templates</b>	<b>13</b>
5.1	Simple Template-Based Data Entry Form . . . . .	13
5.2	Command formats . . . . .	13
5.3	Specifying variables . . . . .	15
5.4	Commands only relevant in workspaces . . . . .	16
5.5	Data entry fields . . . . .	17
5.6	Linking fields . . . . .	19
5.7	Additional web page formatting . . . . .	20
5.8	Advanced formatting options . . . . .	21
<b>6</b>	<b>CIVET Workspaces</b>	<b>23</b>
6.1	Workspace Management . . . . .	24
6.2	User-specified annotation vocabulary using <b>category</b> . . . . .	25
6.3	Automatic annotation/skip editing mode only: . . . . .	26
6.4	Additional information on categories . . . . .	27
<b>7</b>	<b>Annotation and Editing Collections</b>	<b>29</b>
7.1	Comments on annotation and editing . . . . .	31
<b>8</b>	<b>Coding and Text Extraction</b>	<b>33</b>
8.1	Note on deleting texts . . . . .	35
<b>9</b>	<b>Preferences</b>	<b>37</b>
9.1	Programming note . . . . .	38
<b>10</b>	<b>Projected Features</b>	<b>39</b>

<b>11 Appendix 1: Sample Template File</b>	<b>41</b>
<b>12 Appendix 2: Input Format</b>	<b>43</b>
12.1 Collection fields . . . . .	43
12.2 Category fields . . . . .	43
12.3 Text fields . . . . .	43
12.4 Case fields . . . . .	44
12.5 Date formats . . . . .	45
12.6 UTF-8 Encodings . . . . .	45
12.7 Sample File . . . . .	45
<b>13 Appendix 3: Supporting Files and Source Code Settings</b>	<b>47</b>
13.1 Files in /static/djciv_data . . . . .	47
13.2 Additional settings that can be changed in civet_settings.py . . . . .	47
13.3 Documentation . . . . .	48
<b>14 Appendix 4: Prototype on Google Application Engine</b>	<b>49</b>

## Preface

This is the documentation for the CIVET—Contentious Incident Variable Entry Template—data entry system. CIVET was developed under 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](mailto: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](#). This documentation is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License; CIVET is open source and under the MIT license. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.



## INTRODUCTION

This is the documentation for CIVET <sup>1</sup>—Contentious Incident Variable Entry Template—customizable data entry system. CIVET was 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 had 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 <sup>2</sup> 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 still 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. These are primarily in the “Workspace Management” page.

<sup>1</sup> <http://en.wikipedia.org/wiki/CIVET>

<sup>2</sup> An earlier prototype was implemented in the Flask framework: see Appendix 4

## 1.2 Status of the Program: 31 August 2015

The NSF funding for the project ended on this date. At this point, all of the documented features of the program should be working except as noted above. However, we are just beginning the process of operational field testing and it is likely—which is to say, inevitable—that some additional bugs will be found, hence this is still considered “Beta-0.9” rather than “1.0.” We currently have two field tests underway, and are hoping to get some additional ones going, and will be posting bug fixes to GitHub promptly as these appear and are resolved.

At present, we have not developed any software for generating the workspace files, though we expect to have at least a couple programs available in the next few months. The problem here is that identifying the various metadata and components in a set of texts is highly specific to the text source, and to date we’ve not found general solutions for this. As noted in the final chapter of this document, over the next year or so we will be seeking additional funding for tool development for these “front-end” tasks, though given the very slow pace of the public funding cycle, this is unlikely to occur until late in 2016 at the earliest. In the meantime, we will be leveraging tools developed in existing projects and, of course, would very much appreciate the contribution of any ancillary tools that the user community develops, particularly for common sources such as Lexis-Nexis, Factiva, ProQuest, LDC Gigaword, and various news feeds and social media.

## 1.3 Status of the Program: August 2016

Over the past year, CIVET has been used in two projects, though both with my assistance in generating the YAML files and with some additional customization, much of which has been incorporated into the general system.

To my knowledge, however—and if you know of some project using this, please let me know—it has not been used for the originally intended purpose of providing a platform which would allow someone with relatively limited programming knowledge to create a web-based form. I suspect this is due to one or more of the following factors:

- The process of getting Django installed, while thoroughly documented, apparently can require some experimentation and tweaking. That said, one of the projects decided to install Django on laptops used by the coders rather than through a server: this allowed the coders to work anywhere.
- When using workspaces, which allow access to the most sophisticated parts of the system, the texts must be converted to the YAML format, a task for which I’ve yet to see a general solution and almost certainly requires Python, perl or Java programming skills
- Realistically, there are only a small number of new projects starting in any given year which are sufficiently large that existing tools such as spreadsheets or Google Forms are inadequate. And many of those, of course, will have resources to directly develop customized pages rather than working within the constraints of CIVET

So, this is essentially just an open-source version of a codebase that I can customize to generate coding forms. Which I’m realizing is probably pretty much what about 95% of open-source projects are, though that still gives the client a whole lot more knowledge and power than they have with a proprietary system, even if they never change a line of code. Whatever.

Some random observations from those two deployments:

- Additional customization of the code has gone quite smoothly, even allowing for the inevitable decay of my comprehension of the system. Granted, even when I’ve forgotten the details of the code I’m still working with my programming idioms, but it does seem like as a base, this is quite solid. The same can be said for the YAML workspace format.
- Neither of the two installations made any use of the manual annotation, and in the second, the `NEVER_ANNOTATE` option was added to bypass this entirely: annotation is either handled automatically using vocabulary files, or directly putting the HTML into the YAML files.



- More generally, though completely expected, the deployment have generated a number of interesting ideas for new features that did not emerge in the abstract design phase. I've also left in some custom code—commented-out or otherwise deactivated— that could be used for examples of further possible extensions.
- On two occasions over the past year there were changes to Django that required minor changes—one or two lines of code— to CIVET in order to keep it running. Unsurprisingly, we have also found that Django is more likely to be fully compatible with up-to-date hardware and operating systems: in particular, one project ran into some issues running it on some old copies of Windows. Once again, this is less of a turn-key system than I'd hoped.

## 1.4 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/>; <sup>3</sup> a PDF version can be downloaded by clicking the `Download PDF` link on the home page. <sup>4</sup>

---

<sup>3</sup> 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.

<sup>4</sup> 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/`



## INSTALLING CIVET

To date we've only installed the system on Macintosh computers (OS-X), 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>

At this point you should see the CIVET home screen <sup>1</sup>

### 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`.

---

<sup>1</sup> If you see a log-in page requesting a user name and password, the log-in requirement has been activated: see the “Authentication” chapter for details on how to use (or deactivate) this.

# 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.

CIVET is a standard [Django](#) application; the documentation below includes instructions for installing the system either locally or in the Amazon Web Services cloud environment. Django is very widely used and has extensive documentation, so your installation requirements are not covered in the CIVET documentation, it should be relatively easy alternative instructions.

Click one of the following links to use the system

[Read coding form](#)

[Read workspace](#)

[Manage workspace](#)

[Set preferences](#)

[Preferences](#)

**Documentation:**

[On-line manual](#)

[Download 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: 31 August 2015

## AUTHENTICATION

Django famously includes, “out of the box”, a very robust system for handling user authentication and permissions. Except for a few minor modifications such as changing the web page headings and providing vaguely informative messages for log-in failures, CIVET simply implements the default version of this, so you can be guided the instructions at <https://docs.djangoproject.com/en/1.8/topics/auth/>.

Authentication is controlled by the *civet\_settings.py* global variable `REQUIRE_LOGIN`. By default, this is set to `True` when `PRODUCTION_MODE = True` and `False` otherwise. If you enter the name of the site without any additional arguments, the program will go to a login page when `civet_settings.REQUIRE_LOGIN = True`; otherwise it will go directly to the home page. Attempting to access the home page when `REQUIRE_LOGIN = True` without a login will redirect to the login page.

### 3.1 Creating a superuser

To keep things simple, CIVET handles the administration of users through the controls available to a “superuser”. To create a superuser, at the *djciv\_site* level of the directory, use the terminal command <sup>1</sup>

```
python manage.py createsuperuser
```

In development mode, start the system with the usual command

```
python manage.py runserver
```

and enter the URL

```
http://127.0.0.1:8000/admin/
```

You should see a page similar to this: <sup>2</sup>

The *Add* and *Change* buttons provide access to a rich set of options for adding users and editing information about them. Clicking on “Users” will give a screen listing all of the users in the system, and clicking on a user name on that screen goes to a page with information about the user. You can delete one or more users from these screens; the “Groups” option allows users to be organized into groups.

### 3.2 Additional notes

1. Accessing the page without additional arguments automatically does a logout.



---

<sup>1</sup> A description of the process can be found at <https://docs.djangoproject.com/en/1.8/intro/tutorial02/>

<sup>2</sup> The options seen in the tutorial version of this screen which allow the editing of the databases have been deactivated since the database structure is tightly linked to various functions of the program, particularly the reading and writing of the workspace files. These could, of course, be modified, but this will need to be done in the program itself, not simply by adding fields.

## CIVET Administration

### Site administration

Authentication and Authorization	
Groups	 Add  Change
Users	 Add  Change

2. Django provides an extensive set of utilities for resetting passwords: for the sake of simplicity. as well as removing a possible venue for mischief, these have not been activated: it should be relatively simple to do this if you would like to have that capability.

At the present time, the AWS deployment does not show the pretty form, but all of the options are still there and function: this will be corrected at some future date.

3. The GitHub version of the program is populated with at least the following:

Superuser: civet-super Password: je-kiffe-grenouilles <sup>3</sup>

User: ima-coder Password: code-code-code!

For the sake of security, you will probably want to delete these after you create your own environment, or at least change the passwords.

---

<sup>3</sup> You were expecting “password”, “CHANGEME” or “12345678”??

## 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.

**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 including downloading the coded data as a tab-delimited file, editing the meta-data, and adding comments to the file. [Beta 0.9: only the data download is implemented; editing the meta-data can be done in a text editor.]

**Set preferences:** This goes to a page where various program preferences can be set manually.

### Documentation

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

**Download PDF** This downloads a PDF file with the documentation.

**Log out** Log out the current user. You will only see this option if log-ins are required: see the chapter on “Authentication.”

## 4.1 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. <sup>1</sup>

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

---

<sup>1</sup> These files are named `CIVET.demo.template.txt` and `CIVET.extract.demo.zip` in the directory `djcvet_site/djcv_data/static/djcv_data/` and can be modified there.

## CIVET: Select template file

Select a file:  No file chosen

Coder [optional]:

[Download](#) template demonstration file

Fig. 4.1: CIVET file selection screen



## 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.<sup>1</sup> 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.

### 5.1 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.

### 5.2 Command formats

Commands generally have the following format

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

---

<sup>1</sup> 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:

[".txt" suffix will automatically be added]

## 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.<sup>2</sup> 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.

### 5.2.1 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;. <sup>3</sup> 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

---

<sup>2</sup> This feature is actually a bit more subtle: cancellation is invoked when the first character in a line is “-” and there is a “:” in the line, which will always occur with a command. This allows default texts such as “- - -” to be used, though a default text such as “- - -: we really want to confuse CIVET” will cause an error.

<sup>3</sup> 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.

## 5.2.2 Errors in template commands

There is a fair amount of error trapping as the commands are processed; any problems will be 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.

## 5.3 Specifying variables

### 5.3.1 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, the output depends on the preference `civet_settings.USE_TEXT_FOR_MISSING`, which also can be set on the “Preferences” page. If this is True, the text will be used; otherwise the string in `civet_settings.MISSING_VALUE` will be used.

**Example:**

```
save:
worldregion, eyewitness, groupname, comments

save:
worldregion [regioncode], eyewitness, groupname[], comments

save:
worldregion, eyewitness, groupname, groupname [groupcode], comments
```

### 5.3.2 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]
```

### 5.3.3 filename

Sets the default file name for the downloads: this can be changed before downloading.

**filename:** page-text

**Example:**

```
filename: our_wonderful_data.csv
```

### 5.3.4 Special save variables

`_coder_` Coder text entered in the *CIVET template selection* page

`_date_` Current date. this is currently in the form YYYY-MM-DD. <sup>4</sup>

`_time_` Current time in hh:mm:ss format

## 5.4 Commands only relevant in workspaces

### 5.4.1 discard

Sets an initially unchecked checkbox for the special variable “`_discard_`”, which can be used to indicate that a collection has been evaluated by a coder but nothing was coded. When this is checked, a case is generated for the collection containing only the “`_discard_`” variable; those cases are not used to generate data.

**discard:** entry-title

**Example:**

```
discard:  Texts are not codeable
```

### 5.4.2 comments

Creates a textarea box for the special variable `_comments_` which will be added to the “casecmt” meta-data for the case being coded. `_comments_` can also be added to the output data like any other variable, but this is not required. The default size of the text box is 4 x 64 characters; alternative sizes can be specified by adding an empty set of brackets followed by `rows` and `cols` using the same format as the `textarea` command. <sup>5</sup>

**comments:** entry-title

**comments:** entry-title [] rows = number cols = number

**Example:**

```
comments:  Enter any additional comments about this case
```

### 5.4.3 header

Sets the HTML code for the display of collection information at the top of the editing and coding screens. The text of `field-name` will be substituted for the optional token `_text_` in `HTML-text`

**header:** HTML-text [field-name]

`field name` should be one of the following

**workspace** Workspace file name

**collection** Collection ID (`collid` field)

**comments** Collection comments (`colcmt` field)

The three fields are displayed in this order; they default to null strings. The individual `header` commands must be separated by blank lines; otherwise, consistent with the command syntax, <sup>6</sup> the latter lines will be ignored.

---

<sup>4</sup> This format can be changed in the function `get_special_var(avar)` in `civet_form.py`: It is specified using the extensive Python/C date format operators shown [here](#).

<sup>5</sup> In fact, the `comments:` command is just a shorthand for `textarea: entry-text [_comments_]`, and this will have the same effect, with the contents added to the metadata.

<sup>6</sup> Neither a bug nor a feature: just is what it is.

**Example:**

```
header: <h3><span style="color:blue">Workspace _text_ </span></h3>' [workspace]
header: <b>Collection:</b> _text_ ' [collection]
```

### 5.4.4 Special save variables for workspaces

These variables will not include any texts that were deleted using `shift-click` on the lede.<sup>7</sup>

**\_collection\_** collid field of the collection

**\_publisher\_** Comma-delimited list of the `textpublisher` fields of the texts in the collection

**\_bibliorefs\_** Comma-delimited list of the `textbiblio` fields of the texts in the collection

## 5.5 Data entry fields

Any of these commands can be prefixed with `"/"`, which inserts a `<p></p>` or a `"`, which inserts a `<br>`.

### 5.5.1 *checkbox*: Binary 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:**

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

### 5.5.2 *select*: 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``
```

<sup>7</sup> At present, only these two fields are available, but it is relatively straightforward to add the others by just following the existing code that you locate in a search for “textpublisher”

### 5.5.3 *dynselect*: Select from dynamic pull-down menu [workspaces only]

Pull-down menus which are specified dynamically using the `categories` section in the collections are specified with the syntax

```
dynselect: entry-title [var-name]
category-name
```

**Example:**

```
select: Region [worldregion]
statelist
```

If the category-name is not found in the collection, the control defaults to a single entry `---`: this allows for situations where there were no items found in the category.

### 5.5.4 *radio*: 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
```

### 5.5.5 *textline*: 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>
```

### 5.5.6 *textclass*: 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 four standard classes:

- `nament`: named-entities, which are determined by capitalization

- **geogent:** geographical locations, which are determined by a combination of prepositions and capitalization<sup>8</sup>
- 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>
```

### 5.5.7 *textarea*: 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: Description [descript] rows = 2 cols = 64
Briefly describe the incident
```

## 5.6 Linking fields

### 5.6.1 *link*: Linking *select* and *textline* fields

The **link:** command can be used to connect a **select:** or **dynselect:** menu to a set of **textline:** fields so that their content is filled in from the menu if something is selected, but otherwise these fields can be filled in manually.

**link:** one-to-four-textline-vars [select-var-name]

The list of variables is space-delimited and all of the variables must have been defined before the **link:** command is encountered, but otherwise the command can be anywhere in the file.

The rules for extracting the fields depends on the number of variables specified:

1. Insert the entire menu item, which can have any format, in the text field
2. The entire menu item has the form `xxxxxx [yyy]` The `xxxxxx`—which can contain anything except `[ ]`—is inserted in the first field; `yyy` is inserted in the second field. This is typically used for names and codes, e.g. `Algeria [DZA]`
3. The entire menu item has the form `xxxxxx (zzzz) [yyy]` The `xxxxxx`—which can contain anything except `( )`—is inserted in the first field; `zzzz` is inserted in the *third* field, and `yyy` is inserted in the second field. This is a somewhat awkward generalization of the four-variable option. Example: `Algiers (Algeria) [DZA]`

<sup>8</sup> This is only done in the automatic annotation if `civet_settings.USE_GEOG_MARKUP = True`; see the discussion in the *Preferences* chapter.

4. The entire menu item has the form `xxxxx (lat, lon) [yyy]` The `xxxxx`—which can contain anything except `( )`—is inserted in the first field; `lat` is inserted in the third field, `lon` in the fourth field, and `yyy` is in the second field. This is typically used for names with geocoordinates and codes, e.g. Algiers (36.757, 3.063) [DZA], but `lat` and `lon` do not need to be numbers

At present, there is only minimal error checking to insure that the fields are delimited correctly: These are taken from a set menu, not user input, so it is the responsibility of the form developer to make sure these aren't ambiguous (e.g. the menu option Florence (Firenze) (43.821641, 11.286954) [ITA] will generate the string Firenze) (43.821641 as the `lon` field). Alternatively, you can add code in `civet_coder.html` to accommodate the more complex formats.

```
link: countryname countrycode [countrymenu]
```

```
link: cname ccode clat clong [geolocmenu]
```

## 5.6.2 *textsource*: Extract sources from annotated text

A `textsource`: field can be automatically linked to a `textclass`: field so that the information in the `textid`: and/or `textbiblio`: field for the workspace text block where an annotated word or phrase has been extracted will be automatically added. This linkage is done by using the variable name from a `textclass` field with `_src` added to the end.

Except for the linkage, a `textsource` variable acts like a regular text variable: information can be typed or pasted into the text box—typically this will be done from the Source information that is visible when the Show Comments option is active—and the variable can be saved.

If the variable name does not correspond to a `textclass` variable with an added `_src`, `textsource` behaves the same as `textline`.<sup>9</sup>

The `textid`:/`textbiblio`: content is controlled by the preferences “Use `textid` in source citation:” and “Use `textbiblio` in source citation:”; the default uses only `textbiblio`..

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

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

### Example:

```
textsource: Source for city [cityname_src] width=40  
<enter source>
```

## 5.7 Additional web page formatting

### 5.7.1 *title*: Set page title

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

```
title: page-title
```

### Example:

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

---

<sup>9</sup> In a future version of CIVET we hope to have a facility where citation information can be transferred into a `textsource` field by clicking on a lede but this has been implemented yet. *Preferences* chapter.



### 5.7.2 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.

A “/” in the page-text will add a line-break `<br>`. To include a “/” in the text, use “//”.

**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; this is equivalent to the “/” prefix in the data entry commands.

### 5.7.3 *newline*: Insert a line break

Adds a new line in the form; this is equivalent to the “/” prefix in the data entry commands.

**newline:**

### 5.7.4 *newpage*: Insert a page break

Adds a new page to the form.

**newpage:**

## 5.8 Advanced formatting options

A CIVET form is simply a web page, and consequently can be controlled by the standard commands for displaying web pages, notably cascading style sheets (CSS).

### 5.8.1 Set css

Adds the text which follows the command to the `<style>...</style>` section in the page head. The text block is terminated by a blank line.

**css:**

one or more lines of css definitions

### 5.8.2 Set form division sizes

This is a short-cut that for most options just changes the size of various components in either of these forms:

**size:** [division-name] width = <length> height = <length>

**size:** [division-name] width : <length>; height : <length>

<size> can be any of the CSS “length” formats: [http://www.w3schools.com/cssref/css\\_units.asp](http://www.w3schools.com/cssref/css_units.asp).

*division-name* is one of the following:

**body** over-all size of the page

**civ-editor** size of CKEditor text box on the annotation page

**civ-text-display** size of the scrolling text display on the coding page

**civ-form** size of the coding form created by the template in both the basic form and the coding page

**Notes:**

1. The system does not check for the validity of either the division names or the CSS <length> specification; if they can’t be interpreted they are ignored.
2. *size* commands can occur anywhere and can be combined with a *css* command: if they occur before the *css* command the contents of *css* will override *size*, and vice-versa if they occur afterwards: the CSS string for the <style>...</style> section is assembled in the order the commands occur.
3. Because CSS doesn’t use object-like inheritance, the *size:* [body] command resets *all* of the properties of *body*, leaving only *width* and *height* set in the style, e.g.

```
body {  
    width:900px;  
    height:700px;  
}
```

If you want to change the size but also retain other characteristics, use *css* to define the complete *body* style.

4. The *civ-editor* command changes the size in the configuration of CKEditor rather than any CSS, so this name cannot be used in *css*: (well, it can be used but it won’t do anything...). CKEditor does not allow the “%” option to be used as a “height”: see [http://docs.ckeditor.com/#!/guide/dev\\_size](http://docs.ckeditor.com/#!/guide/dev_size). The CKEditor defaults to the width of the page (more or less) and a height of 200px.
5. The *civ-form* and *civ-text-display* names correspond correspond to <div class=’name’> in the content of the form; you can modify these directly by using a *css:* ``command. The ``*size* command resets the <name>-*size* class, which only controls the size.

### 5.8.3 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 setting `civet_settings.HTML_OK = True`.]

Adds arbitrary HTML code without escaping.

**html:**

one or more lines of HTML

## 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 <sup>1</sup> 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 the chapter *CIVET Coding Form Templates*. 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. <sup>2</sup> 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 <sup>3</sup> 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. <sup>4</sup>

The program is sensitive to file names:

- Any file ending with .yaml is assumed to be a CIVET -formatted collections file

<sup>1</sup> <https://en.wikipedia.org/wiki/YAML>

<sup>2</sup> 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 OS-X 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.

<sup>3</sup> Hahaha...just a little programmer joke...

<sup>4</sup> For example, the error in the variable values string in the example screen 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]','naticid': '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

- 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 [Version 1.0: Not yet implemented—see comments on setting globals in the “Preferences” chapter.]

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.
- The system currently translates UTF-8 encodings to ASCII <sup>5</sup> using the Django function `encoding.smart_str()`. We expect to eventually convert the program to Python 3.x (at present it is Python 2.7) which is utf-8 “native” but it isn’t there yet.

## 6.1 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 0.9, 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

<sup>5</sup> UTF-8 is an expanded *Unicode* character set that includes accented letters, “smart quotes” and many many more characters not found in the older *ASCII* (American Standard Code for Information Interchange) character set. UTF-8 is very widely used on the Web so if you have downloaded texts, there’s a pretty good chance they contain at least some UTF-8 characters.

workspace. The remaining functions will eventually be implemented but in the meantime these tasks can be done using a text editing program.

# 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](#)

## 6.2 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 `name`, `num` or `date`. The program currently accommodates up to 99 categories.<sup>6</sup>

`color` can be any of the 140 named HTML5 colors,<sup>7</sup> 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

<sup>6</sup> 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()`

<sup>7</sup> See [http://www.w3schools.com/html/html\\_colornames.asp](http://www.w3schools.com/html/html_colornames.asp)

two-digit color from the CIVET palette.<sup>8</sup> 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.

## 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

### 6.3 Automatic annotation/skip editing mode only:

When the program is running with `Always apply annotation: True` and `Skip editing: True`,<sup>9</sup> categories can be further visually differentiated using one or more of the following font specifications

- bold: **bold face**
- italic: *italic*
- under: underline

So for example `[Green bold italic]` will display the category in an italic bold font colored green.

---

<sup>8</sup> 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`

<sup>9</sup> This is the configuration typically used when just coding the texts with automated annotation. We plan to retrofit this to the editor as well but adding it to the annotation was a simple hack, and adding it to the editor is a little more complicated.

## 6.4 Additional information on categories

1. Generally, matching of words and phrases is not case sensitive: in the example below, both “killed” and “Killed” will match. However, if the word in the category list is all uppercase—e.g. NATO, IRA, ISIS—it will only match all-uppercase strings: this should deal with most cases of acronyms, in particular US and IS. A word or phrase can only be in 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.<sup>10</sup>

2. 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.

3. 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.<sup>11</sup> This be a text file with one phrase per line and the code in brackets; a line beginning with `#` is treated as a comment.

4. As with texts, UTF-8 encodings are translated to ASCII using the Django function `encoding.smart_str()`.

### Example:

```
category:action [red italic]
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 bold under]
codes.nationstate.txt

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

<sup>10</sup> If you want it now, delete the test `if endx == idx+len(st): in CIVET_utilities.do_string_markup()`.

<sup>11</sup> 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`.

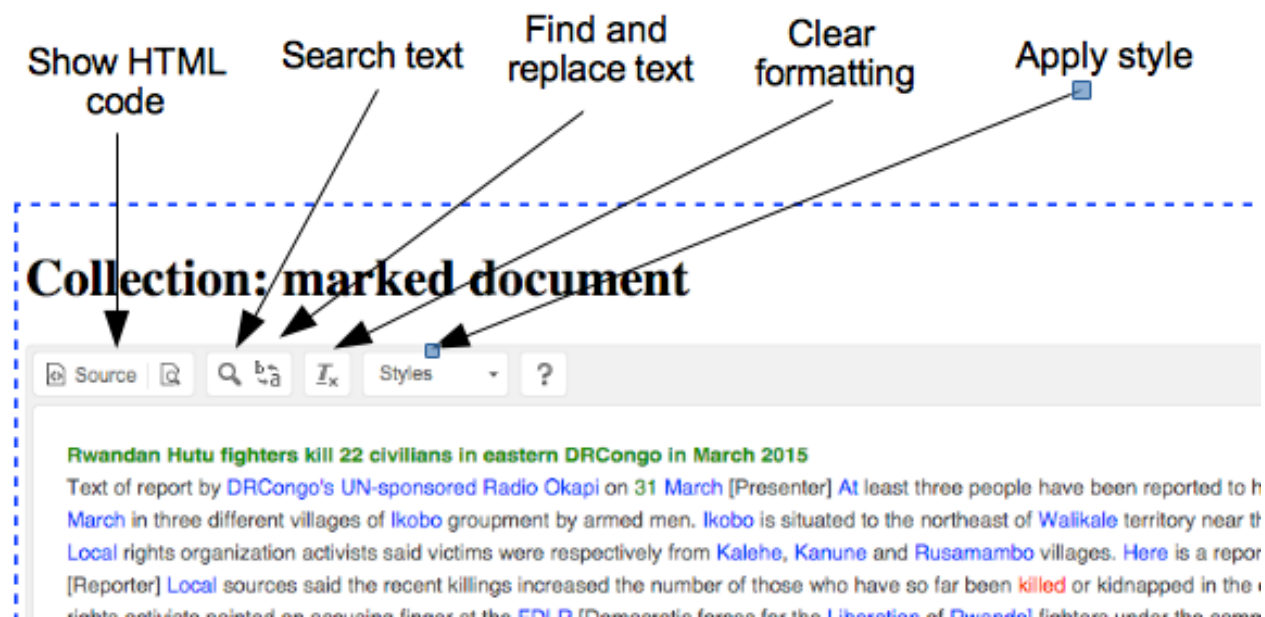




## ANNOTATION AND EDITING COLLECTIONS

The annotation and editing page for workspace collections <sup>1</sup> implements a minimal version <sup>2</sup> of the open source Javascript “CKEditor” <http://ckeditor.com/> 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:

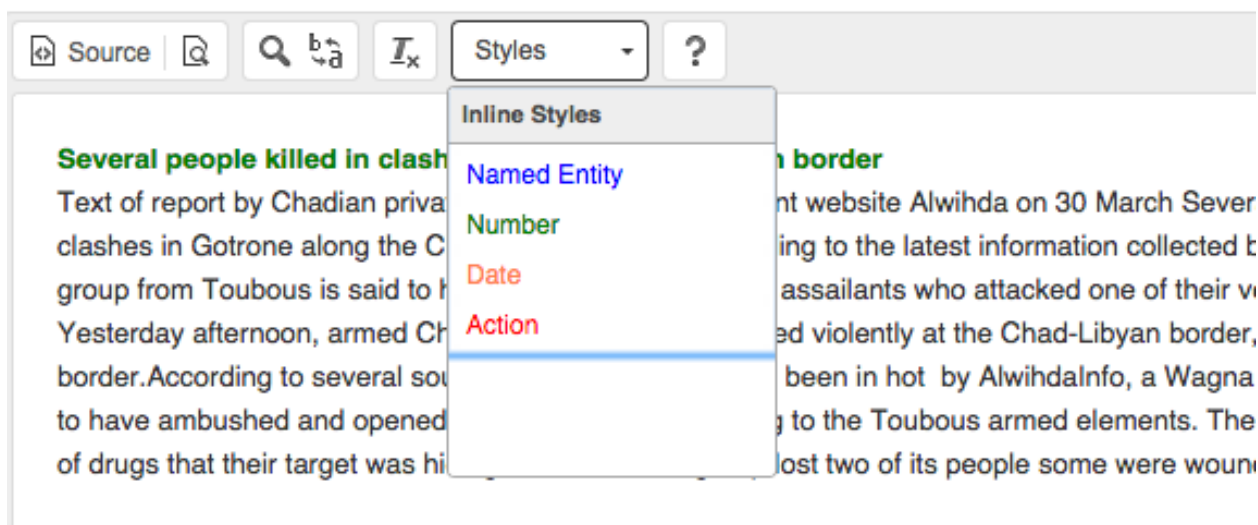
**Named-entities:** This is based on capitalization; consecutive capitalized words are combined.

---

<sup>1</sup> If you are not seeing this screen, `civet_settings.SKIP_EDITING` is probably set to `True`: this can be changed on the “Preferences” screen.

<sup>2</sup> 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.

# Collection: TestTexts\_001



**Location [optional]** When the `Use preposition-based geographical markup` preference is set to `True`, these are named-entities which are preceded somewhere in the text by prepositions in the list `'at', 'to', 'in', 'from'`. See additional discussion in the “Preferences” chapter.

**Numbers:** Digits and numerical words and phrases such as “one” and “two-hundred.”

**User-specified categories:** See the discussion of *categories*

Annotation is done automatically when `Always apply annotation` preference is set to `True`; this can be changed on the “Preferences” screen.

## Save edits and select new collection:

This saves whatever annotation has been done to the internal database<sup>3</sup> 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.

<sup>3</sup> 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.

## 7.1 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 that occurs at the end of a string.
2. A word or phrase can be annotated only once.<sup>4</sup> 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.
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

A local political leader was shot and killed by unknown gunmen.

Fig. 7.1: The category listing:

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

would result in the annotation

A local political leader was shot and killed [4] by unknown gunmen.

Fig. 7.2: whereas category listing:

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

would result in the annotation

A local political leader was shot and killed [1] by unknown gunmen.

Fig. 7.3: 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`<sup>5</sup>

<sup>4</sup> 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.

<sup>5</sup> 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.

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.<sup>6</sup>

---

<sup>6</sup> This list can be extended in the regular expression `pat1` in `civet_utilities.do_NE_markup()`.

## CODING AND TEXT EXTRACTION

The CIVET coding form screen in the demonstration version is shown below.<sup>1</sup>

**Workspace Demonstration file**

Comments: Text file source: Apr15.OTH.stories.txt

Show comments Show all content Hide all content

**Bangladesh: Rights group says 64 killed [2] in police custody, cross-fire [2000-01-01]**

Text of report headlined "ASK report Jan-Mar, 64 killed [2] in 'crossfire', custody" published by Bangladeshi newspaper The Daily Star website on 1 April As many as 64 people were killed [2] in custody of law enforcers and so-called crossfire in the first three [3] months of the year, said rights organization Ain o Salish Kendra (ASK). Of them, 46 people were killed [2] in so-called crossfire - 15 by Rapid Action Battalion (Rab), 24 by police and seven [7] by other law enforcement agencies. The number of people who died in custody between January and March is 18. Ten [10] of them were killed [2] in police and detective firing, according to a statement of the rights watchdog last night. Contacted, Deputy Director Maj Rumman Mahmud of the Rab's Legal and Media Wing said they were yet to see the report and would comment after going through it. The ASK statement was prepared on the basis of reports published in the national dailies and its own findings on incidents of human rights violation.

**Demonstration of CIVET form linked to annotated texts**

Type of incident: ☒ Demonstration ☐ One-sided Violence ☐ Armed Clash

Type of weapons:

If "Other", provide details in the 'Description' section

Was the incident a suicide attack? ☒

Location:

Maximal injuries:

Who was involved:

Brief description of incident:  
Enter brief description here

**Options after saving:** Continue coding this collection Code next collection Select new collection Download workspace and return to home page

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

1. Unless `civet_settings.SHOW_ALL_CONTENT = True`, only the content of the first text will be expanded; to expand or collapse these, click on the lede (green text).<sup>2</sup> The date of the article follows the lede in brackets.

Shift-click on the lede will *delete* the text: the lede and text disappear and from any subsequent codings. The text actually remains in the workspace file until it is permanently removed (or the deletion is reversed) in the workspace management. See the notes below for more details on this operation.

2. There are three controls at the top of the text display:

---

<sup>1</sup> 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.

<sup>2</sup> If you are switching back to the text from a text-extraction box, you will need to double-click: the first click switches the focus to the text; the second toggles the content

- **Show/hide comments:** toggles the display of the comments and sources for each text: these are initially hidden.<sup>3</sup>
- **Show all content:** shows the content for all of the ledes
- **Hide all content:** hides the content for all of the ledes

3. 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

**Who was involved:** people

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

4. 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 that is in an expanded text highlighted using a green background. The arrow keys can be used to move the highlighted text into the field. These operate as follows:

**Right arrow:** Highlight the next text in the category

**Left arrow:** Highlight the previous text in the category

**Down arrow:** *Replace* the contents of the field with the highlighted text.

**Up arrow:** *Append* the contents of the field with the highlighted text. The appended texts are comma-delimited.

If the highlighted text is off the screen, the window will automatically scroll to place the text on the bottom of the screen. If the text contains no words in the category, a pop-up window will alert you to this.

If an annotated category field has an associated source field, that information will be automatically replaced or added when the down or up arrow is used. If a reference is already in the source field and information is being added from the same source, this will not be repeated. References can also be added to source fields using copy-and-paste.

**Note:** If there are a number of phrases in the target category—this occurs frequently for the named-entity and geographical-entity categories—and the phrase you want to extract is not in the first expanded block, click on the ledes to collapse them until you get to a text that does contain the target phrase. If the earlier ledes collapsed, the first phrase highlighted will be in the expanded lede, so you will not need to hit the right-arrow key many times to highlight and extract it.

5. Copy-and-paste from the text to the data fields work as you would expect; text can also be entered and edited manually.
6. If bracketed values are included in the string, the system takes the value from within a set of brackets that is the final item<sup>4</sup> in the phrase: earlier sets are assumed to be part of the text. For example, the value of the phrase Islamic State [ISIS] [mnsa] will be “mnsa”; the value of the phrase Islamic State [ISIS] militia will be “Islamic State [ISIS] militia”.
7. 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.

---

<sup>3</sup> If the `textcmt` field for the text block was empty, the display will show `Comment: ----`. If the `textbiblio` field for the text block was empty, no `Source:` line will be shown.

<sup>4</sup> Specifically, the system checks whether the final character in the string that is not whitespace is ‘]’. The output when the system is expecting to find a bracketed value and does not is controlled by the preference `civet_settings.USE_TEXT_FOR_MISSING` which can be changed on the “Preferences” screen.

**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.

## 8.1 Note on deleting texts

Deleting a text changes the value of the `textdelete` field to `True`: the text remains in the workspace file but will not be displayed again. Deletion also generates a case with the standard `casedate` and `casecoder` fields and the following fields in the `casevalues` dictionary

```
_delete_ : True
_textid_ : textid for the deleted text
```

This can be used to track the deletion of specific texts. version Beta-0.9 does not have any internal utilities for using this information but those functions may be added in a later version.

Deletion is tracked through the hidden text field `deletelist` in *civet\_coder.html*.





---

## PREFERENCES

This page has standard HTML check-boxes for setting the status of some of the variables affecting the work flow and initial presentation of the texts.

**Note:** The “Default” values are those in the “off-the-shelf” version of the program: if you are using a version that has been customized for your specific project, these may have been changed. And if so, further changing them may have unpredictable consequences for the proper functioning of the program.

**Always apply annotation** Always apply automatic annotation to texts that have not been previously annotated.

- Default: `True`
- “civet\_settings.py” variable: `ALWAYS_ANNOTATE`

**Never apply annotation** Never apply automatic annotation to texts: this is used when the annotation has already been done in the YAML file. When `True`, the “Code next collection” button in the coding screen will read the next collection then display the text with the form without any additional markup.

- Default: `False`
- “civet\_settings.py” variable: `NEVER_ANNOTATE`

**Show all content in coder** In the coder, initially expand the content of all of the ledes.

- Default: Only expand the first lede.
- “civet\_settings.py” variable: `SHOW_ALL_CONTENT`

**Skip editing** When reading a collection, skip the editing screen and go directly to the coder: this is typically used when dealing with texts that have already been annotated or where the form does not have any fields that use annotation.

When combined with `Always apply annotation: True`, the “Code next collection” button in the coding screen will read the next collection, apply the automatic annotation, and display the annotated text with the form. In this mode, the automatic annotation is not saved.

- Default: `False`
- “civet\_settings.py” variable: `SKIP_EDITING`

**Use text if value is missing:** This controls the output when `save` specifies a value output and a bracketed value is not the final element of the text string. If `True`, the text will be used; otherwise the `MISSING_VALUE` string will be used.

When combined with `Always apply annotation: True`, the “Code next collection” button in the coding screen will read the next collection, apply the automatic annotation, and display the annotated text with the form. In this mode, the automatic annotation is not saved.

- Default: `True`
- “civet\_settings.py” variable: `USE_TEXT_FOR_MISSING`

**Use preposition-based geographical markup:** Use prepositions to attempt to identify named entities that are geographical locations: capitalized phrases that are preceded by the prepositions in the list `'at', 'to', 'in', 'from'` are assigned the category “Location”. If a phrase is identified *anywhere* in the text as a possible locations, all instances will be labelled with that category; that label will take precedence over the standard “Named entity” category.

- Default: `False`
- “civet\_settings.py” variable: `USE_GEOG_MARKUP`
- “civet\_settings.py” preposition list: `GEOG_PREPOSITIONS`

**Use textid in source citation:**

**Use textbiblio in source citation:** These control the content of the `Source:` that is saved in a `textsource` command and displayed in the `Comments:` `textid` and `textbiblio` refer to the fields in the texts in a workspace file. When both are true, the source has the form “textid:textbiblio” where the content of the field is substituted for the name, unless `textbiblio` is empty, in which case it has the form “textid”. If only one is true, only the contents of that field are included; if both are false, the source is empty and not shown.

- **Default:** `textid: False`  
`textbiblio: True`
- “civet\_settings.py” variables: `USE_TEXTID_IN_SOURCE`, `USE_TEXTBIBLIO_IN_SOURCE`

**Missing value** Sets the missing value.

- Default: `*`
- “civet\_settings.py” variable: `MISSING_VALUE`

## 9.1 Programming note

We eventually expect to implement an option for setting initial preferences through a configuration file in the workspace, but in the meantime the default values of various global variables are set in the file `civet_settings.py` and should be reasonably well documented there; in most cases these take the values `True` or `False`; those values are case-sensitive.

The preferences page is implemented through those global variables, a very minimal Django form class `PrefsForm` in `forms.py`, and the `set_preferences()` and `get_preferences()` functions in `civet_settings.py`. If you wish to make additional global variables modifiable from this screen, you will probably be able to customize it just by following the examples in the existing code.

## 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

## APPENDIX 1: SAMPLE TEMPLATE FILE

```
# CIVET template demonstration file

title: CIVET basic form demonstration

h1:Ministry of Magic Hogwarts Incident Report

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

//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

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

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

//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 any additional information

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

This produces the form

## Ministry of Magic Hogwarts Incident Report

House where incident occurred: ☐ Gryffindor ☐ Hufflepuff ☐ Ravenclaw ☒ Slytherin

Nature of incident

If "Other", provide details in the report section

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

Name of student(s)

Brief description of incident

Reporting official

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

Options after saving:

## APPENDIX 2: INPUT FORMAT

Fields marked with **\*\*** are required. Text fields are limited to 100 characters except for:

- `textoriginal`, `textmkup` and `casevalues` have no length limitation
- the three comment fields `colcmt`, `textcmt` and `casecmt` can be up to 500 characters
- coder ID, which is limited to 32 characters.

### 12.1 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]

**colcmt** collection comments

**categories [optional]** categories and items for dynamic selection menus (**dynselect**)

**texts** one or more related texts

**cases** zero or more coded records

### 12.2 Category fields

These are all indented: the first line is the category name followed by a required colon (:). This is followed by the menu options, one per line preceded by an indent and a hyphen-space (- ``). If the menu option begins with an asterisk (\*``) it is the default value for the menu. The following figure shows an example of menu items specified for three categories, `statecat`, `“torgcat“` and `loccat`.

### 12.3 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.

```
categories:
  statecat:
    - *LIBYA [LBY]
    - UNITED STATES OF AMERICA [USA]
    - IRAN [IRN]
    - SYRIA [SYR]
    - UNITED KINGDOM [GBR]
  torgcat:
    - *ANSAR AL-SHARIA (LIBYA) [2625]
    - SEPTEMBER 11 [2428]
  loccat:
    - *Benghazi (32.11486, 20.06859) [LBY]
    - Melilla (35.29369, -2.93833) [ESP]
    - Baghdad (33.34058, 44.40088) [IRQ]
    - London (51.50853, -0.12574) [GBR]
```

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

**textdelete:** Boolean: text has been marked for deletion.

**textpublisher** publisher [any string]

**textpubid** publisher ID [any string]

**textbiblio** bibliographic citation

**textgeoloc** 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 or manually

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

**textmkupcoder** coder ID

## 12.4 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

## 12.5 Date formats

[This has not been consistently implemented in Beta-0.9]

Dates are ISO-8601 ([http://en.wikipedia.org/wiki/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

## 12.6 UTF-8 Encodings

The system currently translates UTF-8 encodings to ASCII using the Django function `encoding.smart_str()`. We expect to eventually convert CIVET to Python 3.x (at present it is Python 2.7) which is UTF-8 “native” but it isn’t there yet, so you are best off doing your own conversions during the process of converting the original texts to the YAML formatting.

## 12.7 Sample File

The following figure shows an example of a simple YAML file; This is a screen capture of a file being edited with *BEdit*, 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
colledid: 2015-06-08
colcmt: Text file source: Apr15.OTH.stories.txt

texts:

- textid: TestTexts_002_001
  textdate: 2000-01-01
  textpublisher: BBC Monitoring Africa
  textpubid: BBCAP
  textbiblio: BBCAP00020150401eb4100105
  textgeoloc:
  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
```

## APPENDIX 3: SUPPORTING FILES AND SOURCE CODE SETTINGS

### 13.1 Files in `/static/djciv_data`

#### 13.1.1 Files that can be modified using a text editor

**CIVET.demon.template.txt:** Demonstration template file for simple coding

**CIVET.workspace.demon.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

#### 13.1.2 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.

#### 13.1.3 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

### 13.2 Additional settings that can be changed in `civet_settings.py`

These are global options but do not appear in the “Preferences” menu since there is little reason to change them dynamically (and usually plenty of reasons not to)

**Hide ‘Read coding form’** Hide the ‘Read coding form’ button on the home screen: this can be used if you only intend coders to be using workspaces.

- Default: `False`
- **`civet_settings.py` variable:** `HIDE_READ_CODING_FORM`

**Hide ‘Read workspace’** Hide the ‘Read workspace’ button on the home screen: this can be used if you only intend coders to be using coding forms.

- Default: `False`
- **`civet_settings.py` variable:** `HIDE_READ_WORKSPACE`

**Hide ‘Preferences’** Hide the ‘Preferences’ button on the home screen: this can be used if you don’t want coders changing these.

- Default: `False`
- `civet_settings.py` variable: `HIDE_PREFERENCES`

## 13.3 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.

Because of the images and the redundant files in the “\_build” directory, “`djcivet_site/docs`” is a very large directory, about 70% of the size of the full “`djcivet_site`” directory, and can be removed in deployments where you are just planning to use the on-line documentation. In that situation, the roughly 1 Mb file “`djcivet_site/djciv_data/static/djciv_data/civdocs.pdf`” can also be removed, to the code for the system itself is only about 2 Mb.

## APPENDIX 4: PROTOTYPE ON GOOGLE APPLICATION ENGINE

An earlier demonstration version of the program, written in the Flask <http://flask.pocoo.org/> micro-framework, is deployed as an application on the Google App Engine at <http://ace-element-88313.appspot.com/>. The code for this version can be downloaded from <https://github.com/philip-schrodt/CIVET-Flask>. The Flask version has most of the data entry commands, but none of the workspace commands.

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.

If you don’t need the content management or extraction facilities of CIVET workspaces, Flask is simpler and easier to deploy than Django, but has much the same model-view-controller logical structure, and like Django uses the “jinja2” template system for web pages. It should be relatively straightforward to retro-fit the new features of the forms system—notably the “/” and “/” prefixes—to the older code.