**COONEY — PhiloLogic**

<4 images>

**PhiloLogic4 and the Android PhiloReader Apps: Toward Building a Full-Featured PhiloLogic API**

Cooney, C., Gladstone, C., Shandruk, W., Morrissey, R., ARTFL Project, Roe, G. and the Australian National University

The ARTFL Project would like to submit a proposal for a short paper on the development of an API for PhiloLogic4, our next-generation corpus query and text retrieval platform for digital humanities databases, and to demonstrate Android PhiloReader Apps that we are building to interact with that API.1

A primary goal of the PhiloLogic4 project has been to allow ARTFL or any digital humanities group using this software to develop a variety of results display and user interfaces with ease.2 For example, the databases ARTFL has already released in beta form under PhiloLogic4 for traditional browser access have features such as frequency sidebars for query results, links within those sidebars for faceted browsing, and dynamic Time Series reports.3 ARTFL’s PhiloReader Apps extend and exemplify this fundamental design goal. They take advantage of PhiloLogic4’s simple set of query parameters and flexible results object formatting to enable text search and retrieval on handheld devices.

ARTFL intends these apps to serve as a lighter-weight alternative to web browser apps for interacting with the PhiloLogic4 installations of text databases on our main production servers. The interface has been designed with a focus on the reading functionality that PhiloLogic4 already offers in its web incarnation. Users can conduct word and/or metadata search from a toggling drawer with the aim of finding and reading text sections.4 Word search results can be returned in concordance and frequency reports. Any metadata value can serve as a frequency report option, though the most common are word use by title, by date, by author, and even by speaker in collections of plays. More intensive text analysis would require the search form accessible through a web browser.

Functionally, the Android code interacts with PhiloLogic4 databases simply by sending search queries and then displaying the results that PhiloLogic4 sends back over the network. Those results, like all the data the PhiloLogic4 API passes both internally and externally, are formatted as JSON objects. From the search terms the user enters, the app builds a query URI compliant with the PhiloLogic4 API, which includes parameters such as number of results per page and metadata values, as well as a parameter that calls specifically for a JSON object (‘&format=json’). For basic concordance and bibliographic searches, the URI currently points at the main script PhiloLogic4 uses to handle all interaction from web browsers (‘dispatcher.py’).5 For example, a basic concordance query for ‘sun’ from the Shakespeare app has this URI:

http://artflsrv02.uchicago.edu/philologic4/shakespeare\_demo/dispatcher.py?report=concordance&q=sun&method=proxy&title=&start=1&end=25&pagenum=25&format=json

For the web version of PhiloLogic4, the dispatcher directs calls for secondary queries, like frequency searches and table of contents requests, to scripts that run behind the scenes. In order to minimize extra coding on the server side, the app calls those CGI scripts directly for these same queries. A frequency by title search for ‘sun’ from the app points to the frequency generation code to get PhiloLogic4’s internal JSON object:

http://artflsrv02.uchicago.edu/philologic4/shakespeare\_demo/scripts/get\_frequency.py?report=concordance &q=sun&method=proxy&title=&frequency\_field=title&format=json

The JSON object of a concordance result, for example, can contain chunks of the search result, bibliographic metadata, and a PhiloLogic id used for linking into larger sections of the text. The Android code renders the JSON into a string array and displays search results in a listview.6 The user can then select individual list items to get larger text sections. The Android code submits a second query to the PhiloLogic4 database for that specific text object which is, again, returned as JSON and also contains navigational links to the previous and next sections of the text. This full-text JSON is rendered for the user to read inside a webview.7 We use a webview to display text objects in order to apply the same CSS formatting rules that we use for web versions. The PhiloReader also allows users to bookmark text objects for easy access in later sessions by retaining its PhiloLogic object pointer.

At the time of writing this proposal, we have succeeded in developing a functional API and Android code to work with it as proof of concept. Going forward, we will continue to work to make the API as generic and simple to use as possible, and fully RESTful compliant.

Our ultimate goal is to allow other development teams to pick and choose any subset of PhiloLogic4 functionality through the API to build their own interfaces. For instance, developers could integrate concordance search functionality into a traditional desktop app under Windows or MacOS, or into a web environment like Drupal or Django. Developers could also use the API to plug frequency or collocation reports into modern visualization tools like d3.js. And since the API always returns pointers to original text objects, any new interface will be able to have links from results display into fuller document context.

ARTFL chose Android Java as the initial development language for the apps because of existing in-house capability. At the time of the conference, we will make available a skeleton version of our Android code for other groups to adapt or simply examine to see how we interact with the API. In the coming months we intend to have parallel apps developed for iPads, though we are not certain they will be ready in time to present at DH. Nevertheless, ARTFL believes these Android apps demonstrate the ease and flexibility of using the PhiloLogic4 API to develop new ways of interacting with text databases.

**Screenshots**

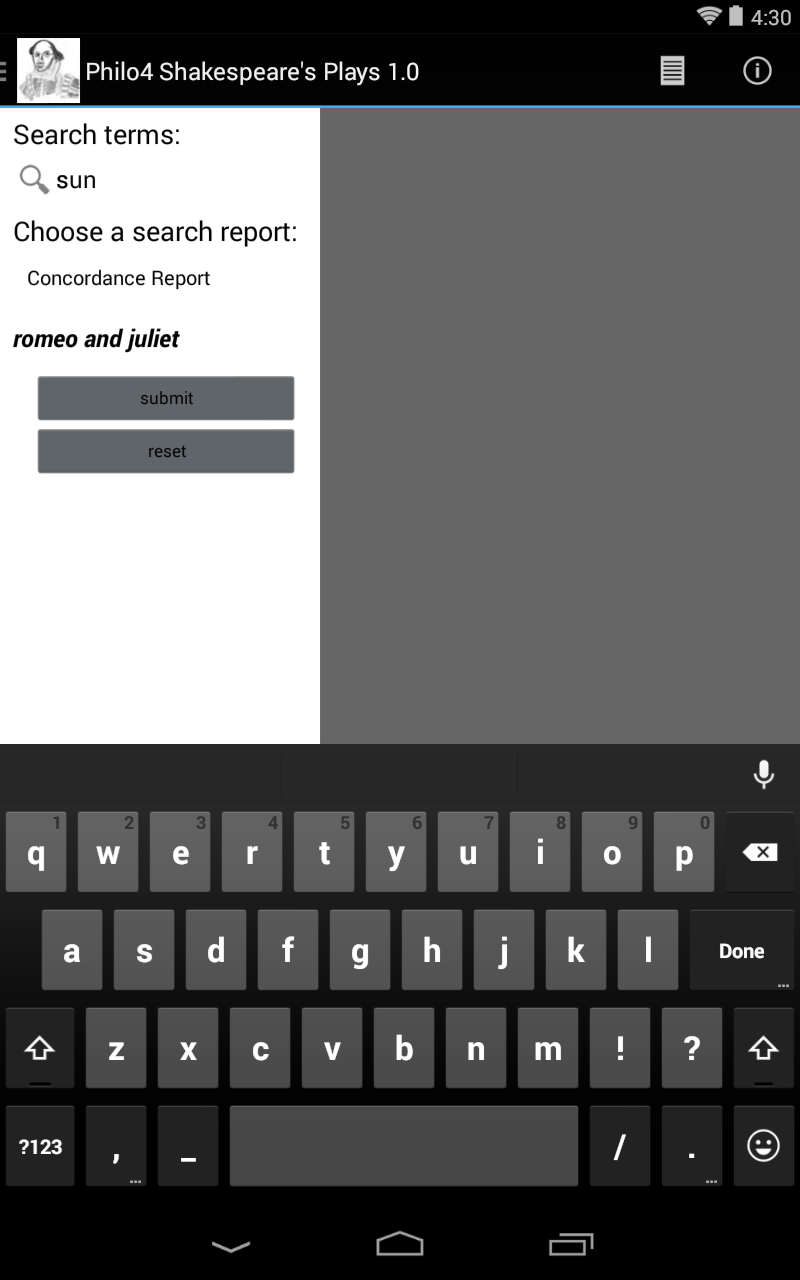


Figure 1. Search for term ‘sun’ in *Romeo and Juliet* in the Shakespeare database.

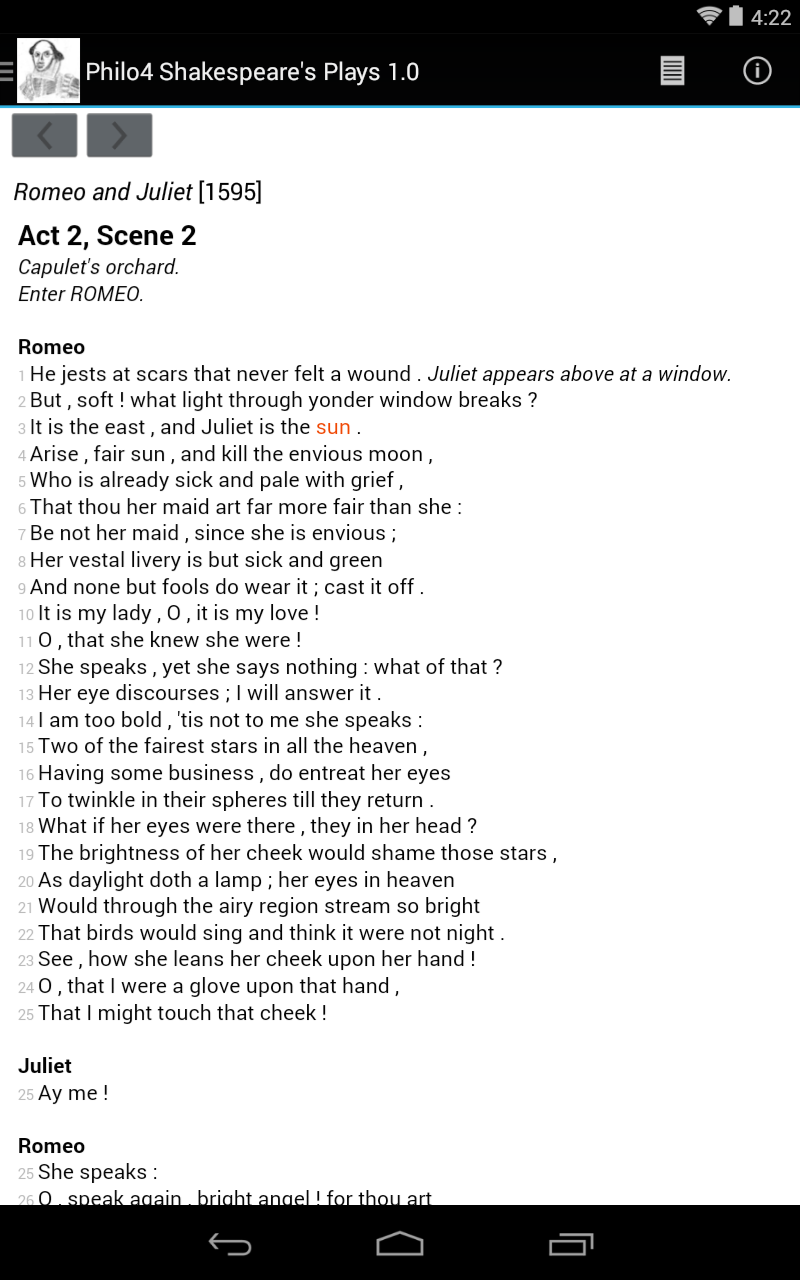


Figure 2. Full-text view from *Romeo and Juliet* with search term ‘sun’ highlighted.

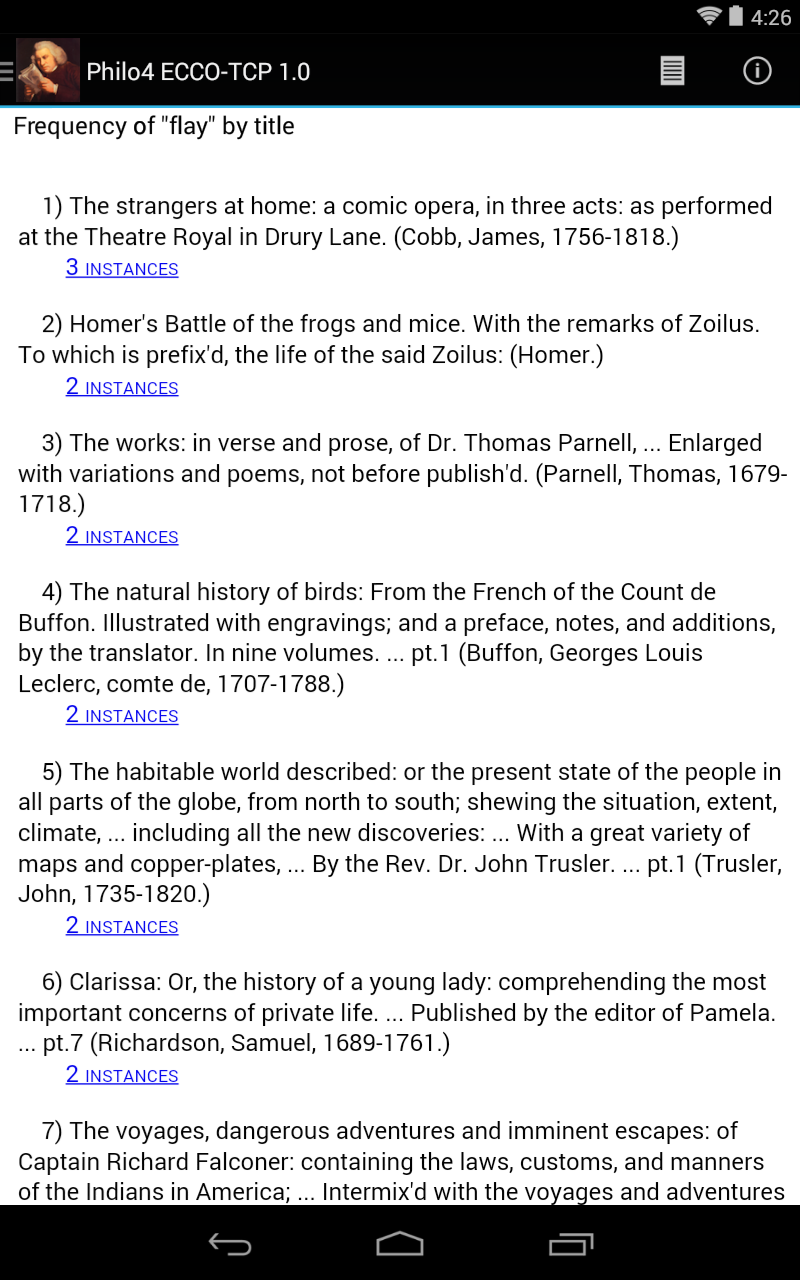


Figure 3. Frequency by title search results for ‘flay’ in ECCOTCP.

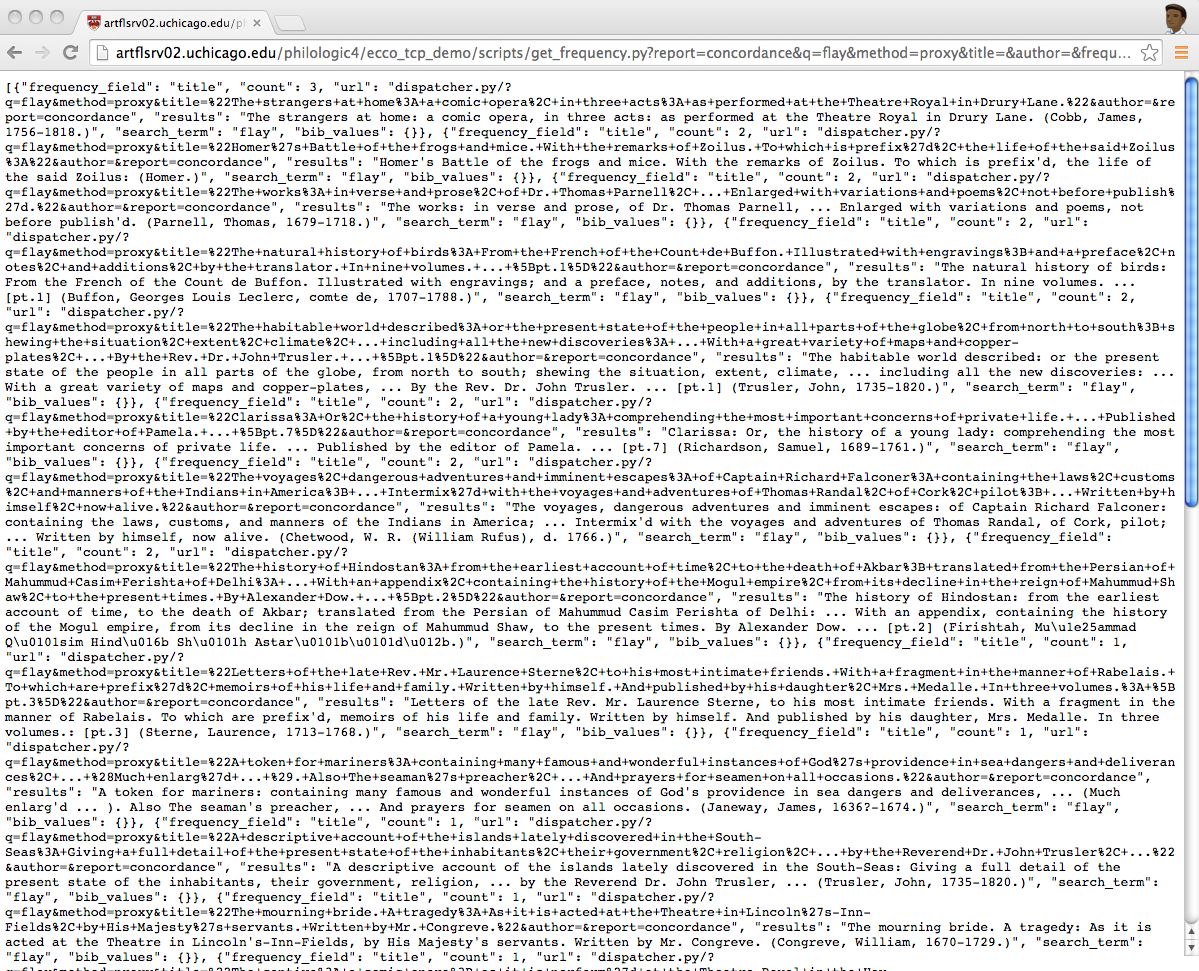


Figure 4. JSON object of frequency by title search results for ‘flay’ in ECCOTCP, http://artflsrv02.uchicago.edu/philologic4/ecco\_tcp\_demo/scripts/get\_frequency.py?report=concor dance&q=flay&method=proxy&title=&author=&frequency\_field=title&format=json.

**Notes**

1. At the time of writing, we have built demonstration apps around the ECCOTCP text collection and the MONK project’s Shakespeare’s Plays data. Versions of these apps can be downloaded from http://artflsrv02.uchicago.edu/downloads/app\_download/.

2. For general background on PhiloLogic4, see Allen, T., Gladstone, C. and Whaling, R., PhiloLogic4: An Abstract Query TEI System, *Journal of the Text Encoding Initiative,* 5 (June 2013). Development is ongoing; code resides at https://github.com/ARTFLProject/ PhiloLogic4.

3. See http://artflproject.uchicago.edu/.

4. See the ‘Screenshots’ section for images of the interface and an example of a PhiloLogic4 JSON object.

5. These calls that the app makes to the dispatcher script are subject to change as we continue to refine the API. Eventually, we might choose to bypass the dispatcher, as we do for frequency and table of contents calls from the app, and instead communicate with CGI scripts exclusively.

6. ListView is Android terminology for a layout that displays a vertically scrollable list. See http://developer.android.com/guide/topics/ui/layout/listview.html.

7. Again, Android terminology for a view that displays web pages. See http://developer.android.com/reference/android/webkit/WebView.html.