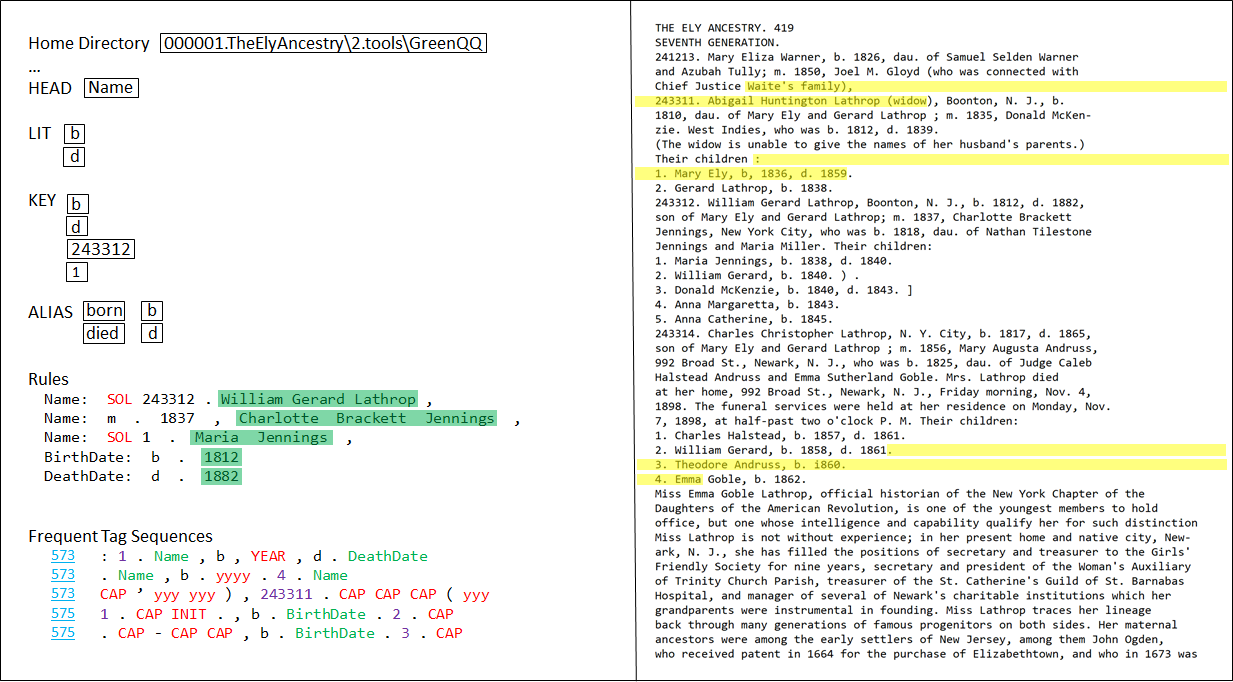
Mock-ups for GreenQQ Interface Development

The interface consists of four levels of user sophistication:

1. Super User Developer: Complete direct control over the underlying csv input specification file.
2. Advanced Developer: Directly set parameters and develop rules using simple user actions.
3. Regular Developer: Indirectly set parameters and develop rules in a form filling motif.
4. Worker User: While extracting information to add to FamilyTree from some page or two of a book, an invisible agent uses the extracted information to configure GreenQQ for extracting information from the entire book. )

Advanced Developer Interface



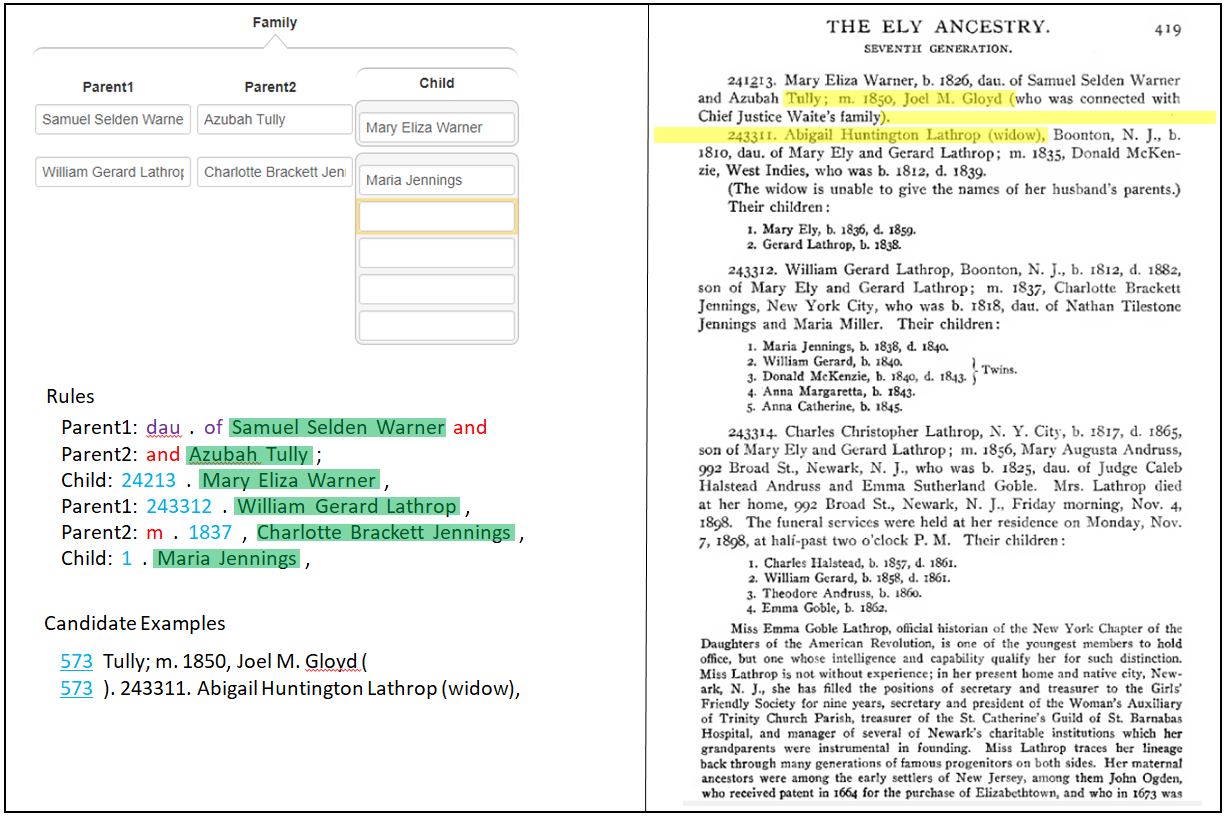
(Note: only the first Rule uses a mono-spaced font and has spaces marked to be visible. All text in Rules and Candidate

Examples should be displayed this way, with a better choice for marking spaces.)

User Actions:

1. Type in parameter values. (Home Directory, …)
2. Type in LIT, KEY, and ALIAS values.
3. Create Rules:
4. Click on a Rules pull-down menu to select a class name. (The form will have been selected on entry to the interface. The names of the form fields are the set of possible class names.)
5. Highlight example search phrase text in the page text and click to establish the text version of the underlying tokens as the mock-up shows. (Other COMET user actions for text selection should also be available for selecting text.)
6. Highlight text to be extracted by the search phrase and click. If the extraction text is included within the search phrase text, it is highlighted in green. If not, the extraction text is appended to the right or left of the search phrase text depending on its relative position to the search phrase in the page and is separated by “…” and highlighted in green. (Example: “DeathDate: died … Nov. 4, 1898” where the underline denotes the green highlighting.)
7. Turn Candidate Examples into Rules:
8. Following a run of GreenQQ, Example Candidate text phrases are returned along with their location in the book. For each candidate, the clickable page number and example text are displayed as the mock-up shows. When a page of one of the candidates is displayed in the right-hand panel of the interface the candidate text is highlighted as the mock-up shows.
9. Rule creation proceeds exactly as for creating rules described above. (A rule can be created from any text, not just the highlighted text. In addition to clicking on candidate example page numbers, users can browse forward and backward and can jump to any specified page with COMET-like controls.)

Developer Interface



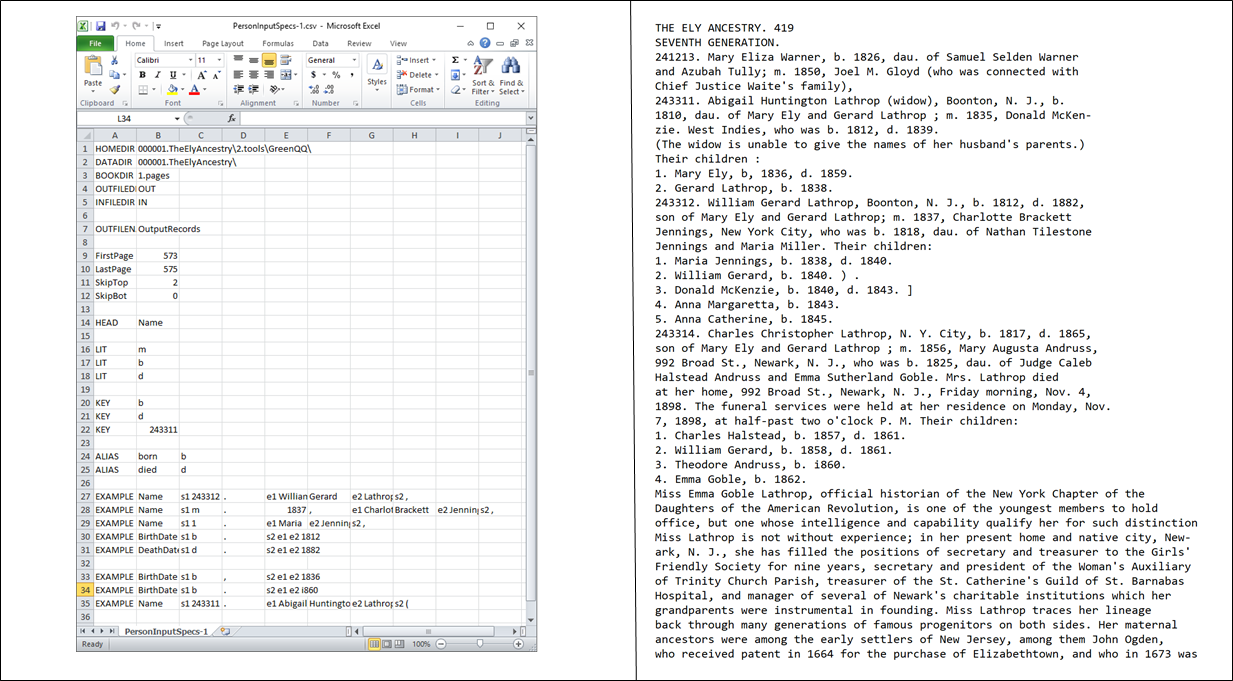
System Actions:

1. Upon form-field fill-in, a rule is generated as follows: The Class name is the form-field name. The Extract is the form-field content. By default, the search phrase consists of a two-text-token prefix of the Extract text, the Extract text, and a one-text-token suffix. (The default is initialized in this way, but can be altered in the Advanced Developer interface.)
2. Upon a GreenQQ iteration request, the system generates and displays Candidate Examples and highlights the text of the examples, if any, in the displayed page. In the mock-up, the left extent from a basis KEY is two text tokens on the left and eight on the right.

User Actions:

1. With COMET-like actions, fill in a form field. (The mock-up is in a state just after having entered Maria Jennings as a Child and then having called for an iteration of GreenQQ.)
2. Adjust Generated Rules (there are three cases):
3. Sometimes the generated Rule is “just right” and there is nothing to adjust. (The Maria Jennings rule in the mock-up is an example, as are the Mary Eliza Warner and the William Gerard Lathrop rules.
4. Sometimes the generated search phrase is too much or too little. In this case the user highlights the desired search phrase in the page and clicks to replace it in the generated rule. (The three rules in the mock-up that were not “just right” to begin with are examples.)
5. Sometimes the desired search phrase does not include the Extract and is thus just plain wrong. In this case the user highlights the desired search phrase and clicks. The system will see that the Extract is not included, determine its proper placement wrt the search phrase (on the left or on the right) and add it. (Example: When the rule “DeathDate: morning , Nov . 4 , 1898 .” for the Person form is replaced by the search phrase “died”, the replacement rule “DeathDate: died … Nov . 4 , 1898” is generated.)
6. Designate literals, keys, and aliases. In the generated rule, a right-click on a text token brings up a menu of possibilities. Selecting “Literal” makes the text token a LIT and colors it red. Selecting “Key” makes the text token a KEY and colors it blue (or if already red, turns it purple – also blue to purple if keys are marked before literals). Selecting “Alias” makes the text an alias of some typed in text. (Once a literal, key, or alias has been marked, it need not be marked again. Thus, in the mock-up, the second “and” and the last three numbers require no user action.)
7. Process Candidate Examples and any other text the same as for all other rules. That is, open a record, fill in the fields, adjust the generated rules, and mark any additional literals, keys, and aliases.

Super Developer Interface



User Actions: Initially, just edit the Excel spreadsheet directly. Eventually, provide some facilities to speed up the editing. Examples: cut & paste page text for initial example search phrases; highlight cells for Extracts; ctrl-click left and right to select search phrases from GreenQQ-generated Candidate Examples; and pre-initialize parameters with defaults. Possibly also, use the Advanced Interface to initialize the spreadsheet.

Worker User Interface



User Actions: The worker is using COMET to gather genealogical information and attach it to FamilyTree from a document that has not been semantically indexed by our extraction engines. The system behind the scenes is looking over the user’s shoulder and gathering information. We can use the gathered information as training data for GreenML and GreenDDA, and we can generate regular-expression extraction rules with GreenFIE. I believe that we can also generate GreenQQ extraction rules. We can then semantically index the book by applying an ensemble of these extraction tools. Thus, we can truly have a “Green” system that improves with use while a user works on completing tasks of interest unrelated to configuring an extraction engine. We can also invite workers to interrupt their information-gathering work and spend some time “programming” the extraction tools (e.g. for GreenQQ by dropping into one of the development interfaces described above).