# The Publishing Accelerator Toolkit (PacTk)

*"Scientific Publishing is a Scientific Discipline"*

Amy Neustein, Ph.D.
CEO and Founder
Linguistic Technology Systems
Fort Lee, NJ
917-817-2184
amy.neustein@verizon.net

***Class Libraries and Full-Text Search/Encoding Tools for Publishing Documents, Datasets, and Multimedia Resources***

## *Overview* ▸▸▸

In contemporary publishing, the standard workflow includes XML-based formats such as JATS or BITS, from which are compiled PDFs and other human-visible documents. The intermediate XML and similar files — those midway in the pipeline — are distinct from files that authors submit directly; they have their own tool stacks and are computationally accessed via protocols such as SAX (Simple API for XML), XQuery, DOM (Document Object Model), and XQSE (XQuery scripting extension). Unfortunately, XML itself is not explicitly built for representing textual data, and examining JATS or BITS files via generic XML technology renders certain publishing-related query and editing tasks difficult to implement. The PDF format is similarly opaque because internal PDF representations of document text include many anomalies due to ligatures, hyphenation, font specs, and other layout-related discrepancies between XML inputs and user-visible output. As a result, internal PDF text only partially corresponds to textual content as understood by authors, editors, and human readers.

Given the limitations of PDF and XML for natural-language documents, numerous competing formats have been developed, providing potential alternatives or extensions to XML. These include TEI (Text Encoding Initiative), TagML (Text-as-Graph Markup Language), BioC (a biomedical text mining standard defined by PubMed Central), Markdown (popular for wikis and social media), and Research Object Bundles (a Semantic Web based specification that covers both text and data sets). With a proliferation of different text-encoding standards, query and traversal protocols that are developed for one format (e.g., JATS and its derivatives) cannot be directly applied to others. For example, the XML parent-child element relationship is replaced in TagML with two different containment styles (logical and extensional), such that DOM and event-based parsers targeted at the simpler XML hierarchy must be reimplemented for the TagML system.

Meanwhile, with regard to PDF, effective search and interaction capabilities depend on supplementing this format's internal natural-language representation with machine-readable text encodings, which might be embedded in PDF files or available as sibling assets. In principle, XML documents could supply such content, although current pipelines are not set up to share JATS/BITS files (for example) in end-user contexts.

From the perspective of PacTk, text encoding evinces a spectrum of distinct *topomorphic regimes* where a proliferation of incompatible document structures prohibits the emergence of common query, traversal, and metadata formats (this nonstandard terminology will be clarified below). The obvious problem is that workflows have to be reconstructed for each incompatible document specification. In practice, such difficulties have almost certainly hindered the adoption of theoretically well-founded formats such as TagML.

The idea behind PacTk is that divergent formats cannot be satisfactorily reconciled simply by proposing yet another format to insert into analogous workflows. Instead, building a more sophisticated and computationally rigorous publishing platform involves constructing integrated computational tableaux weaving together text-encoding, query-evaluation, compiler-theoretic, and front-end concerns.

One important variation amongst text-encoding formats is the level of granularity recognized for textual elements. In JATS, BITS, and most JSON representations, the fundamental discursive units are individual paragraphs. Elements on smaller scales (such as sentences or block quotes) are notated haphazardly at best. LaTeX, for its part, employs reasonable sentence-boundary heuristics — potentially overridden via macros when necessary — but LaTeX sentences are not expressed in a systematic manner (e.g., via auxiliary files) for any purposes other than visual spacing. In TEI and BioC, individual sentences *can* be represented, but this is not mandatory (thus `s` and `sentence` tags are available but not guaranteed to be present in any particular document). By contrast, PacTk is explicitly built around sentence-level units, specifically a "topomorphic sentence-level object model" (TSOM). When all regular text in a natural-language document is explicitly contextualized in delineated sentence objects, many editing, indexing, and user-interface tasks become more tractable (as outlined in the next section).

In contrast to the other formats mentioned here, PacTk is not a single file type but rather a collection of code libraries and classes, with data types representing specific textual elements (sentences, paragraphs, PDF pages, annotations and comment-boxes, etc.). Query and traversal protocols modeled via XQuery, DOM, and so on become expressed by object methods and operators exposed to C++ code, for example (PacTk is thus "topomorphic" — literally meaning "shape of place" — in the sense that navigation between objects is determined by structural properties of divergent document formats, that manifest in competing notions of individuation for structure-sites and any interrelationships declared between them). In short, the PacTk object system synthesizes distinct structuring protocols found in competing formats such as JATS, TagML, and BioC, allowing publishers' workflows to encompass each of these formats (and others) without sacrificing compatibility. PacTk is particularly aimed at publishers who feel constrained by the limitations of JATS or BITS but are reluctant to try alternatives (TagML, BioC, etc.) due to minimal software-ecosystem support and the lack of standardized XQuery and DOM-style interfaces.

PacTk's design also reflects trends in scientific research and the emergence of interdisciplinary fields such as social ecology, translational medicine, and medical humanities. It is the view of PacTk's developers that the value of new investigations often cannot be fully expressed in single-disciplinary terms; nor does one individual publication mark the end of a well-conceived research project. Instead, any single publications are milestones in the unfolding of projects that, in the best case, progress to socially-beneficial deployments. More so than in the past, research projects (summarized but not fully articulated by academic papers) can serve as foundations for scholars' and scientists' technical "identity," potentially more significant than details such as degree specialization or disciplinary affiliation. In recognition of these paradigms, PacTk seeks to help authors build multi-faceted presentations covering data, code, and foundations for practical applications, which authors could build upon for future research work, funding, translational implementation, and then implementational feedback.

The remainder of this document will discuss how PacTk can streamline editing, indexing, and document-preparation tasks, as well as improve User Experience for human readers. Later sections will also address how PacTk may contribute to broader projects wherein text documents are published alongside data sets and/or multimedia presentations (Research Objects/Compendia). Overall, PacTk presents a unified model and coding interface for text queries operating in diverse contexts, including PDF viewers, search engines, document-preparation code, and editing/typesetting. Software employing PacTk to navigate through document objects can leverage interop metadata across multiple formats — such as XML elements and PDF page coordinates — that are typically unavailable in other publishing tools.

### *Features for Editing, Indexing, and PDF Viewers* ▸▸▸

The computational foundation of PacTk is a collection of classes that represented individual sentences and other textual elements in the TSOM family (PDF pages, annotations, footnotes, block quotes, edits or changes across document versions, etc.). In order to initialize these objects, PacTk provides a distinct input format called GTagML, or "grounded" TagML. The term "grounded" reflects how certain structure-sites may be specifically marked as serializing a typed object: i.e., tags, attributes, or text nodes in the neighborhood of a site supply the information necessary to initialize an object-instance during deserialization. In other respects GTagML follows most of the principles advocated by TagML, with novel extensions; for example, GTagML encourages the exploitation of Unicode Private Use areas to disambiguate visibly similar but logically distinct characters/glyphs (e.g., periods may embody punctuation, acronyms, or abbreviations).

In source files, GTagML is loosely based on Markdown, so authors familiar with Markdown from more informal contexts (social networking sites, chat rooms, GIT documentation, etc.) will recognize many of the GTagML styling commands. GTagML sources might be provided by authors directly, or derived from author inputs by digital copy editors or others on the manuscript pipeline. To support GTagML, PacTk provides a new C++ parsing library that is readily extensible (more so than parsers for XML, JSON, and so forth), allowing publishers to customize the GTagML format with new tags, syntax, and data structures specific to their in-house styles and disciplinary focus areas (e.g., specialized tags/data for different scientific fields — chemistry, physics, medicine, etc.; or whatever subject areas are addressed by given publishers, journals, and book series). Alternatively, TSOM objects could be compiled from sources such as JATS or BITS directly.

For workflows that feature JATS, BITS, or LaTeX, TSOM objects may be employed as adjunct data sources through which edits, queries, document version-comparisons, and other computational artifacts are routed. PacTk data can then be mapped both to archiving formats (like XML) and camera-ready generators (like LaTeX), but with a common TSOM background these two different representational facets may be integrated (for example, XML elements mapped to corresponding PDF page coordinates). In effect, PacTk serves to triangulate human-visible presentations (such as PDF) with behind-the-scenes JATS-style formats. Analyses that are currently implemented against XML structures may be more efficiently coded via PacTk objects, with concrete benefits in areas such as copy-editing and indexing.

## ▷ PacTk for Editing

PacTk has an advanced query framework to help find specific locations in text where edits might be warranted (e.g., to comport with journal or book-series conventions) and to correctly review and process such edits. Here are some features:

**Contextual Filters for Text Locations** Often it is useful to restrict the scope of text searches to content with specific presentation characteristics (italics, alternative typefaces, citations), discursive roles (inline/block quotes, footnote text, section/subsection titles), or semantic significance (acronyms, annotations, proper names and place names).

Implementing such filters via XPath/tag restrictions can be inaccurate, because relevant filter criteria could be orthogonal to DOM hierarchies. For example, XML or LaTeX code might wish to wrap elements such as footnotes or block quotes in custom environments, to ensure desired pre- or post-processing. Material that is functionally footnote or quoted text, therefore, may not necessarily be indicated with those specific tag names. The PacTk system allows programmers to define lists of boolean parameters that apply to text-sequences and then activate those flags automatically in specific tag contexts.

In general, PacTk supports queries where ordinary textual restrictions may be augmented with filters for a wide variety of layout and discursive contexts. Such queries could then be used for editing tasks that should be localized to specific contexts, or where a user wants to find a specific piece of text known to have a given context.

## Sentence-Level Queries

Siting queries relative to sentence boundaries can improve precision and express query rationales more succinctly (when compared to paragraph-based organization). For instance, the proper punctuation around a word or phrase is often determined by whether that content appears at the beginning, middle, or end of a sentence. Some query expressions might similarly depend on stipulating that two or more search targets (e.g., two proper names) occur in the same sentence.

Because **PacTk** recognizes almost all text as contained inside sentence objects, queries dependent on sentence boundaries can be directly evaluated on **TSOM** data (such boundaries are unambiguously notated via input sources, rather than detected approximatively using Natural Language Processing). **PacTk** likewise models "topomorphic" navigational relations among sentences, such as between adjacent siblings of parent objects (typically paragraphs) and also nestings wherein a multi-sentence footnote, for example, occurs in a parent-sentence context, or where a block quotes functions in part as a logical child of the sentence that introduces it.

## PDF Page Coordinates

Human readers post-publication — plus authors and copy editors beforehand — typically view documents in **PDF** format for normal reading, even if edits/changes are made on intermediary files. As a result, a bitmapped page-image is the primary medium through which users view single publication pages. Such graphics are typically rendered by **PDF** viewers, but other techniques might be used, such as converting page images to **SVG** (illustrated here in Figure 1, where the relevant component is an embedded web engine that communicates with a **PDF** viewer by QWebChannel; so **SVG** events are processed by **C++** code directly, having only a minimal **JavaScript** routing layer). Failure to interoperate these images with structured text formats is a significant hindrance to efficient document preparation and editing.

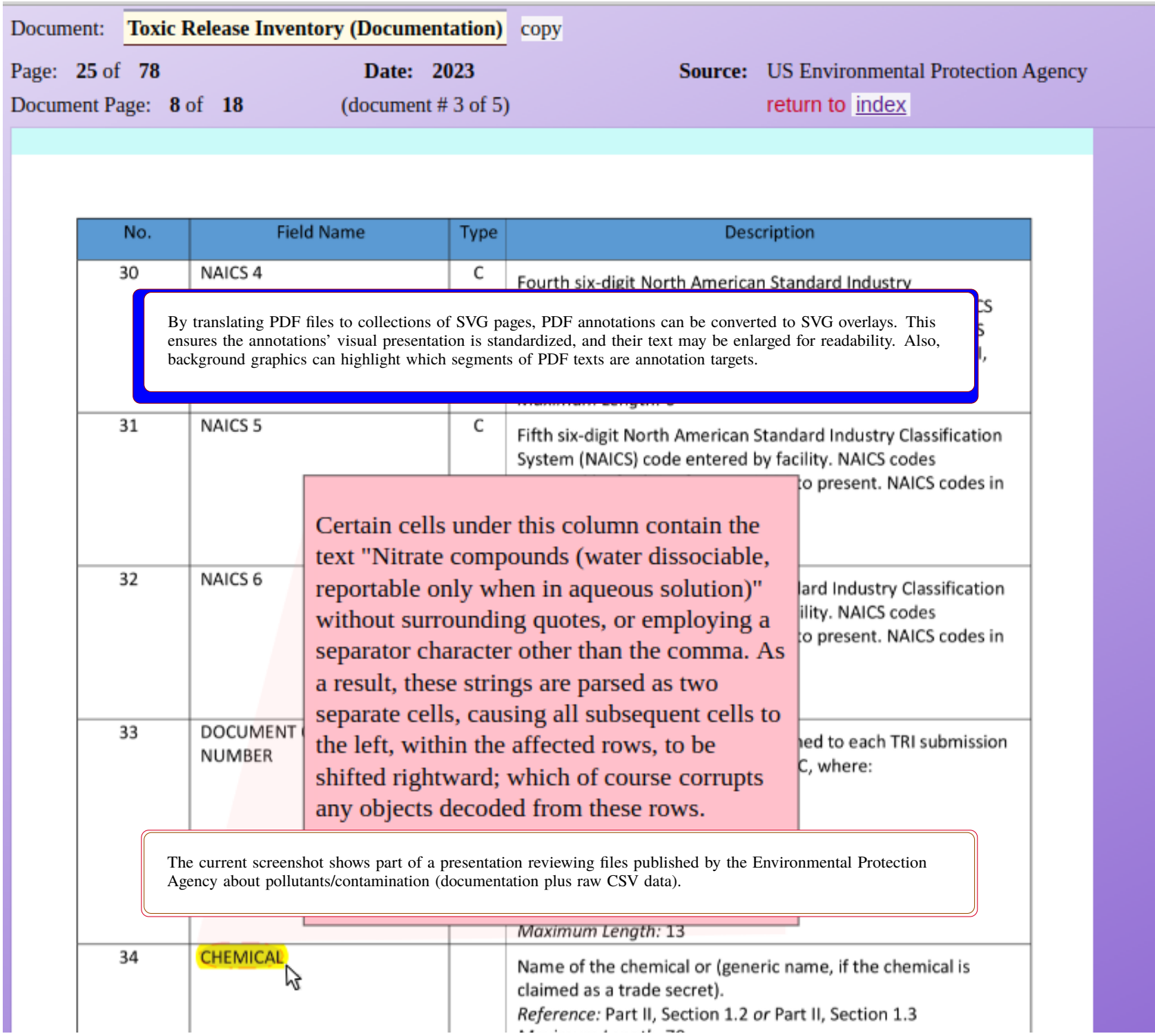



Figure 1: Mapping PDF pages to SVG views for improved annotation support

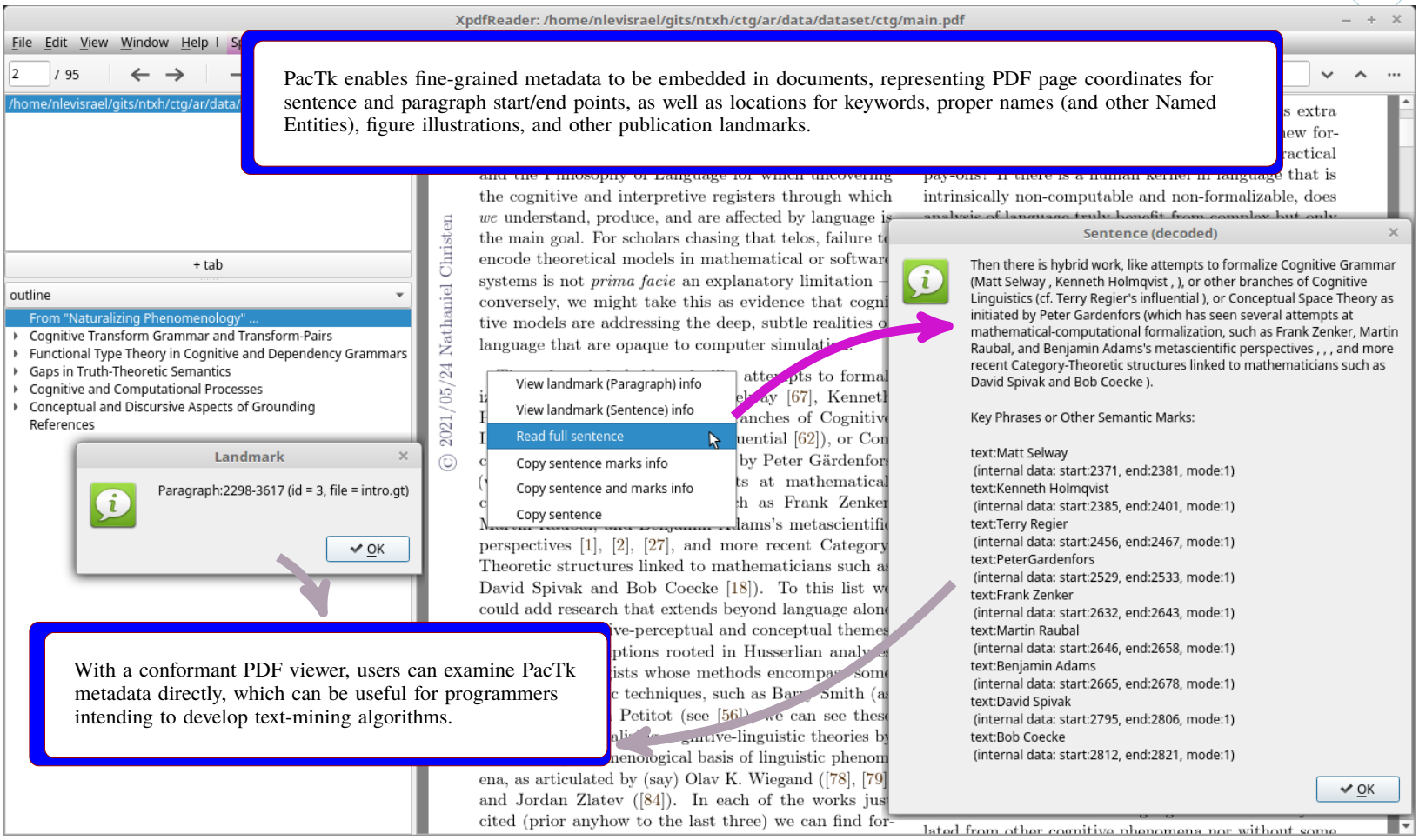



Figure 2: Enhanced GUI capabilities enabled by machine-readable text encoding

By design, **PacTk** supports post-processing algorithms within pipelines whose formats can generate **PDF**-coordinate metadata, such as LaTeX pgfsavepos (and auxiliary write) commands. Page coordinates for significant discursive locations (e.g., sentence and paragraph boundaries, footnote markers, keyphrases and named entities, citations) can thereby be exposed to **PDF** viewers which could accordingly implement novel front-end features such as overlays, context menus, and dialog windows positioned nearby relevant content (Figure 2). For example, a context menu action may include copying the full text of a current sentence to clipboard, based on the screen position where the menu is activated; semitransparent overlays could similarly show the extent of individual sentences during mouseover events.

**PDF Comments and Annotations** In addition to sentence objects, **PacTk** classes model data specifically relevant to the **PDF** front-end milieu, including single pages, links/anchors, and comments/annotations. Objects such as comment-boxes may thereby be leveraged systematically in an editing/review pipeline.

In particular, **PacTk** supports construction of hashtag- or handle-style controlled vocabularies that could annotate **PDF** comments introduced by copy editors or typesetters. Many modifications considered during the editing process follow common patterns, such as proper use of commas around subordinate clauses, or correct formatting of ellipses and bracketed content within inline/block quotes. Assuming that editors consistently include notations for recurring patterns (*insert/delete comma*, *respell word*, etc.), **PacTk** code can filter queries specifically to **PDF** comment boxes which indicate a specific editing category. Index codes within **PDF** comments (automatically inserted by compliant software) may likewise indicate which individual sentence surrounds the annotation, so that the **PDF** view could be cross-referenced against input sources, which facilitates review and processing of manuscript changes.

For similar reasons, **PacTk** supports abstractions such as **XML** custom character entities and LaTeX macros. Many familiar textual components — ellipses, *m*- and *n*-dashes, quotation and footnote marks, etc. — are typically represented in text encoding not via similar visible characters (like two individual dashes or three periods in sequence) but rather through standardized codes or glyphs. The problem is that systematic mappings are lacking when mixing different file types (e.g., LaTeX and **XML**, such as **JATS** or **BITS**) in a single workflow. **PacTk** has a flexible abstraction system that allows character and macro codes to propagate across different generated files.

A related problem is that abstraction codes should generally be opaque to human users in contexts such as comment boxes and copy-paste between software applications. **PacTk** therefore supports declarations for how custom entities/macros and Unicode Private Use Plane (or Area) codepoints should be rendered in plain-text contexts as well as structured formats such as **XML** and LaTeX.

Often users may prefer to initiate or review edits through **PDF** files directly, but in other cases plain-text files might be appropriate instead, without **GUI** overhead. With **PacTk**, for example, authors could receive a pair of text files (or of collections of files indexed, for instance, by chapter) summarizing discrepancies between two manuscript versions. Via sentence-level filtering, such files could be organized such that only sentences that differ from one version to the next are included, with each sentence listed separately. This may be a more convenient

setup for authors to review changes than working through typical GUIs (structured encoding is important because it ensures that version comparisons show valid differences, rather than just anomalies due to divergent font/character styles and line/page breaks or layout). Such breakdown is only possible when the text encoding systematically identifies individual sentences, as with PacTk's TSOM object model.

## ▷ PacTk for Indexing and Search Engines

Authors might use one of several methods to compile an index. They can start with a list of important headings (names, concepts, etc.) and search for relevant passages in the manuscript. Alternatively, authors could read through a document and identify when a particular paragraph mentions a named entity suitable for indexing. Depending on representations or tag sets used (e.g., the JATS string-name element) annotated spans and citations (particularly titles and author names) may also precipitate the provisional heading-list for an index.

Given these various index-preparation methods, it is useful to extend front-end tools with capabilities specific to indexing workflows, such as query matching and manipulating index entry objects. Structured formats like JATS do not provide sufficient computational tractability in themselves. For example, the most popular desktop GUI framework are the Qt C++ libraries, which are the basis of cross-platform PDF viewers such as Poppler and XPDF. PacTk can seamlessly interoperate with Qt, exposing sentence texts as QString objects and expressing GTagML parsing grammars as QRegularExpression instances, paired with lambda callbacks activated by the first matching rule (relative to the current parse-position). More precisely, GTagML has context-sensitive grammars that can include or skip over rule-tests via activated contexts; this allows sections of a GTagML file to utilize syntax different from the default.

With respect to indexing, GTagML has its own notation for index entries (headings, locators, and "see also" style redirections), to serialize indexes in a human-readable format. Correspondingly, PacTk has classes specifically for index entries and associated data. These objects could be initialized from GTagML files or specialized windows floating atop PDF viewers.

Building indexes on the basis of native sentence and index-entry objects has certain benefits as compared to generic (e.g., XML) manuscript formats. These include:

**Multiple Match Strategies** Index locators are typically identified by matching text against entry-headings, but the parameters of these searches will vary by context. Sometimes locators are only relevant when the local sentence includes just the exact heading label, but in other places a desired match could include words in different order (e.g., either first-name/last-name or vice-versa), or a subset (e.g., last name only), or associated concepts with different language entirely (e.g., "The White House" in lieu of a president's name). Concepts can also be grouped into semantic "clusters" (consider *ranson*, *extortion*, *bribery*, *corruption*, *influence peddling*, *lobbying*), with internal similarity-weight matrices. In the presence of a particular index entry, front-end displays can give users options to fine-tune which match strategies are appropriate for query strings seeking locators specific to that entry.

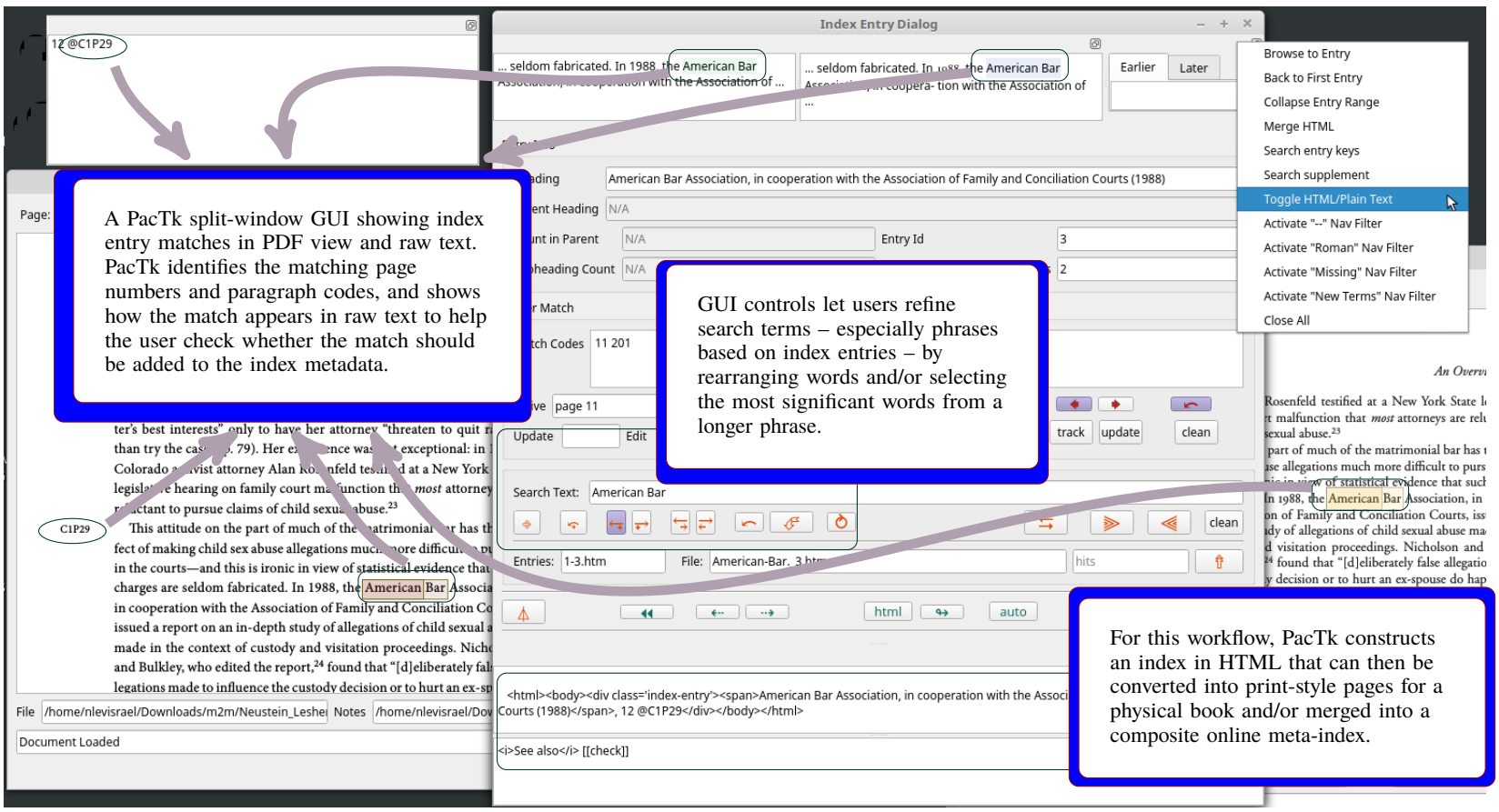



Figure 3: GUI functionality for compiling and curating indexes

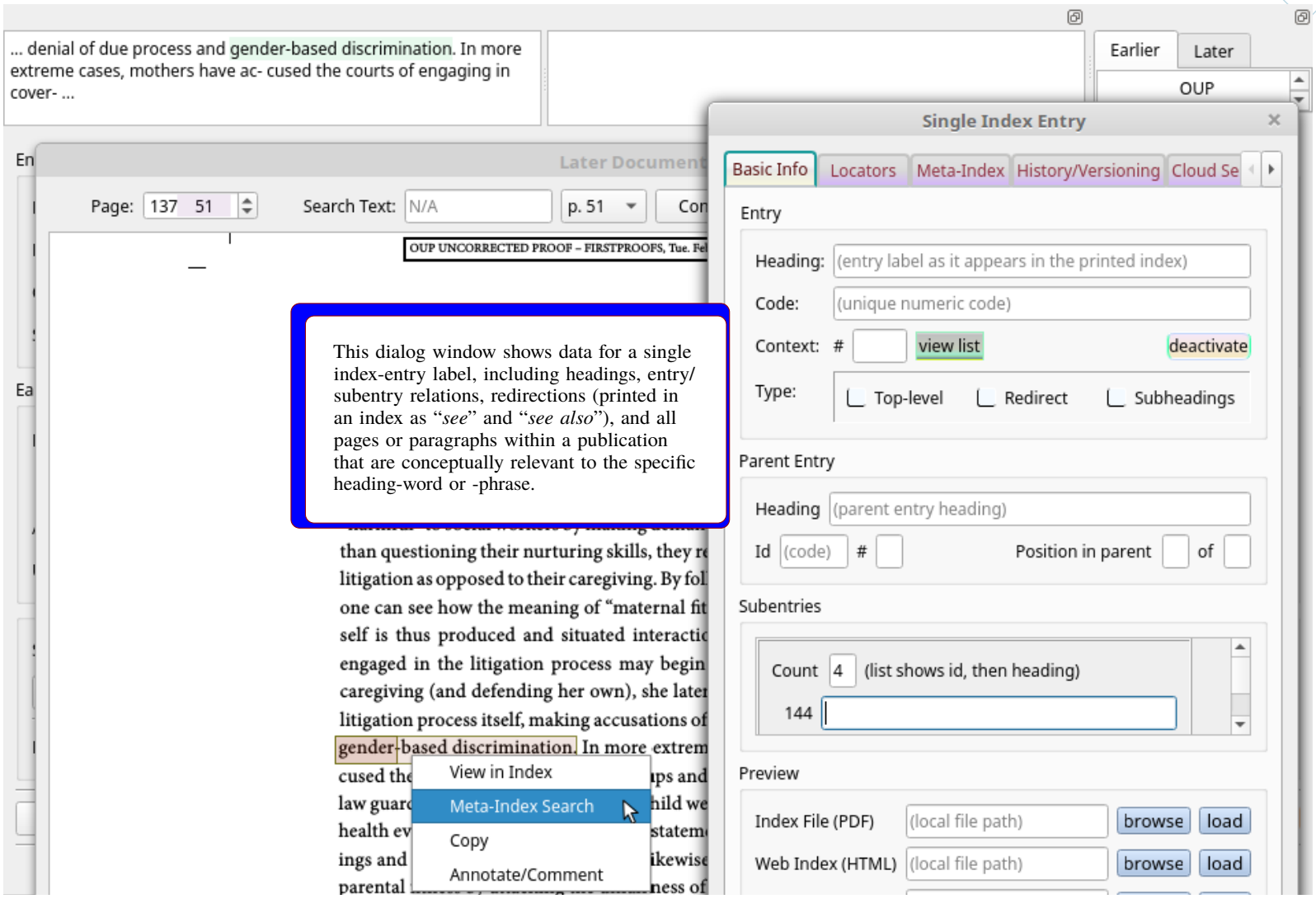



Figure 4: An index entry dialog window (and context menu for single and meta-index views)

**Separating Data from Presentation** Different publishers or book series may choose different visual styles for their indexes. Representing indexes behind the scenes via structured data objects allows them to be curated in abstraction from the specific layout chosen for final publication.

**PDF Context Menus** Authors/editors may prefer to add new index entries or locators directly from PDF pages. PacTk-compliant viewers will read metadata that specifies which paragraphs are in scope at different page-coordinate regions, so paragraph ids for any location can be ascertained from mouse events (e.g., the dropdown point of a context menu).

**Index-Building GUI Components** Index entries are multi-faceted objects that naturally lend themselves to dedicated front-end displays, such as dialog windows (Figures 3 and 4). For example, entries typically include a heading label and then one or more locators (e.g., paragraph ids, replaced eventually with page numbers), then possible subheadings with their own locators, plus "redirectors" (*see* or *see also*) applied either to overall entries or to subheadings. This represents a lot of information to keep track of for each entry, and indexes can include hundreds of entries. Although indexes are eventually shown to readers as ordinary PDF pages (or similar formats), during document prep it is useful to subdivide indexes into individual GUI windows for each entry, with options to navigate between adjacent entries or to traverse on larger scales (e.g., one chapter to another), plus to create new entries on the fly.

**Index-Guided Text Searching** Indexes are curated lists of headings and locators selected by authors/editors for relevance. Whenever a search is posed against a manuscript in any context — e.g., in the query bar of a PDF viewer, or an online search engine, or XQuery-style terms evaluated vis-à-vis JATS files — index metadata can be consulted just in case search terms overlap with headings already included in an index. Such functionality may be augmented by extending index metadata to include concepts related to index entries even if those secondary concepts are not indexed themselves (as with semantic clusters, mentioned above). In effect, index metadata may declare that all locators modeled as relevant for their specific entry are also relevant to some list of associated concepts, such that whenever those latter concepts are targeted in a query the explicit locators are included among the query results (Figure 8).

**Series or Archival Meta-Indexes** Once a standard format is developed for indexes in some document collection, it becomes possible to pool multiple instances into a common archive. Each time a new publication is finalized, all of its index terms could be inserted into a bibliographic database allowing users to search across multiple books/manuscripts in a given book series, journal collection, or document repository. Such meta-indexes can support searches that are more precise and granular than conventional search-engine technologies based on occurrence vectors and stemming/lemmatization alone (Figure 5).

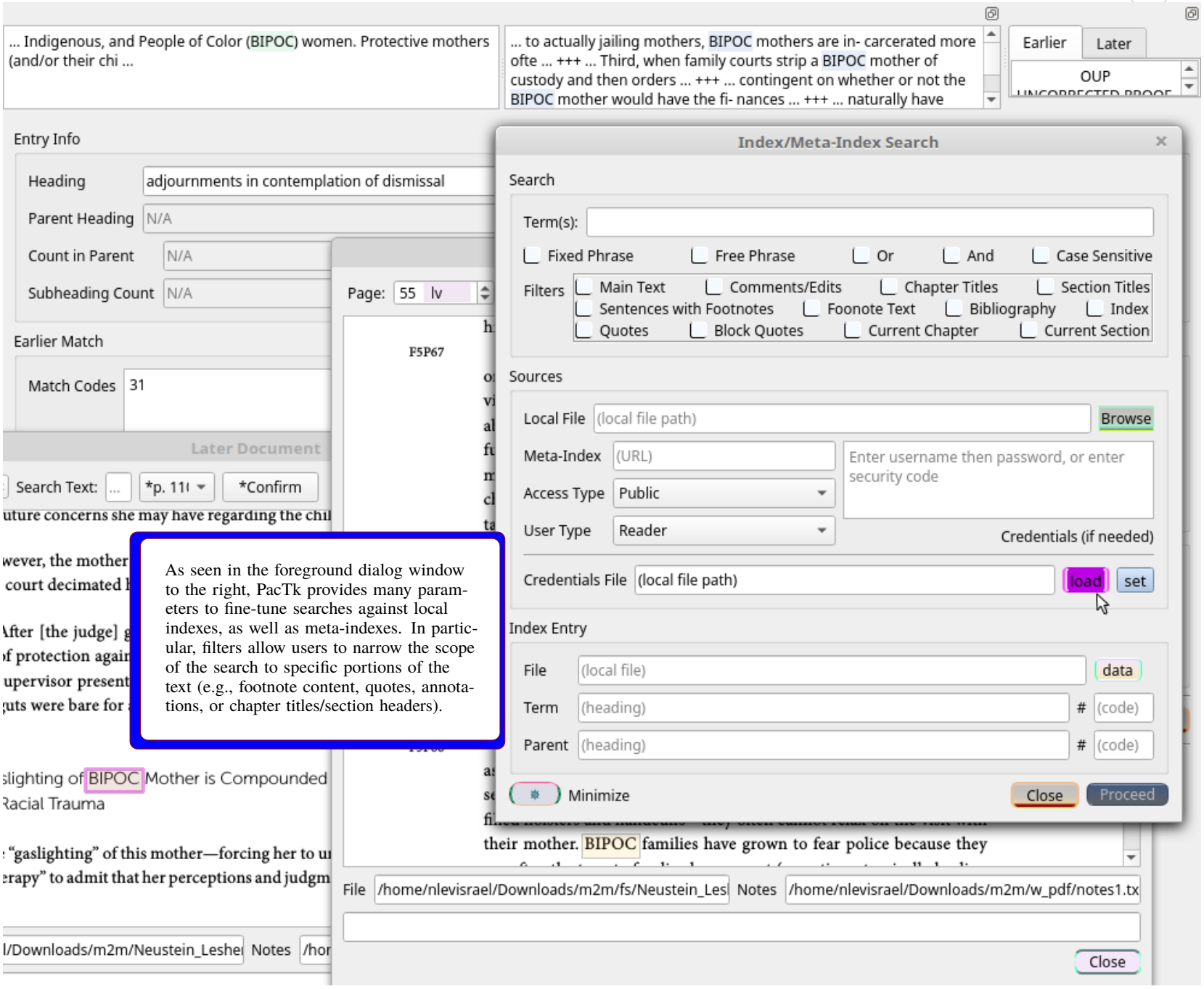



Figure 5: Dialog window for local and meta-index searches

## Academic Search Engines

In online contexts, "indexing" typically refers not to human-visible book indexes but rather to compressed data files that catalog nontrivial words and concepts in a manuscript. Such indexing is typically non-contextual; for example, it is not represented whether a word/phrase is present in text with a special purpose (quotes, footnotes, chapter/section titles, etc.) and sentiment/discursive roles are ignored (e.g., a document might be identified as *about* some topic, but more specific criteria such as arguing *for*, *against*, or neutrally *evaluating* theories or claims are not modeled). Granularity may be limited by the scope of an archive, because efficient queries can only expend a few seconds per document if searches are to be conducted against a large collection.

Nevertheless, search functionality can take steps to increase query specificity via meta-indexes and related metadata. For one thing, users may specifically consult search areas that are restricted in scope — say, a single University Press, book series, repository (PubMed Central, for example), or journal collection. Publishers who curate meta-indexes for mid-size collections can support much more detailed evaluations per document: a book series might have several dozen titles, not tens of thousands. PacTk supports construction of corpora query languages that express granular searches over multiple documents, with similar logic to filtered/contextualized text searches in a single PDF document (Figures 6 and 10). Even where queries are executed via conventional vector-database methods, PacTk allows authors or digital editors to exercise control over how word-occurrence lists are compiled. For example, TSOM objects can include lemmatized word-lists per sentence; authors may then control for circumstances such as two different meanings reduced to the same linguistic stem, which might not be properly ingested by search engines indexing documents through their default algorithms.

Although PacTk's originating focus is publishing — where the primary components of a repository are manuscripts/documents in formats like PDF — this technology could potentially be used in other environments. For instance, in clinical trials and point-of-care software the fundamental assets are health/medical records; but search features, as well as plugins and multimedia (discussed plugins), might be extended to include resources such as diagnostic images, cytometry gating, assay graphics, tumorogenesis simulations, etc. Publishing-related plugin architecture (e.g., document viewers in scientific software) may be adapted and resued for these EHR contexts.

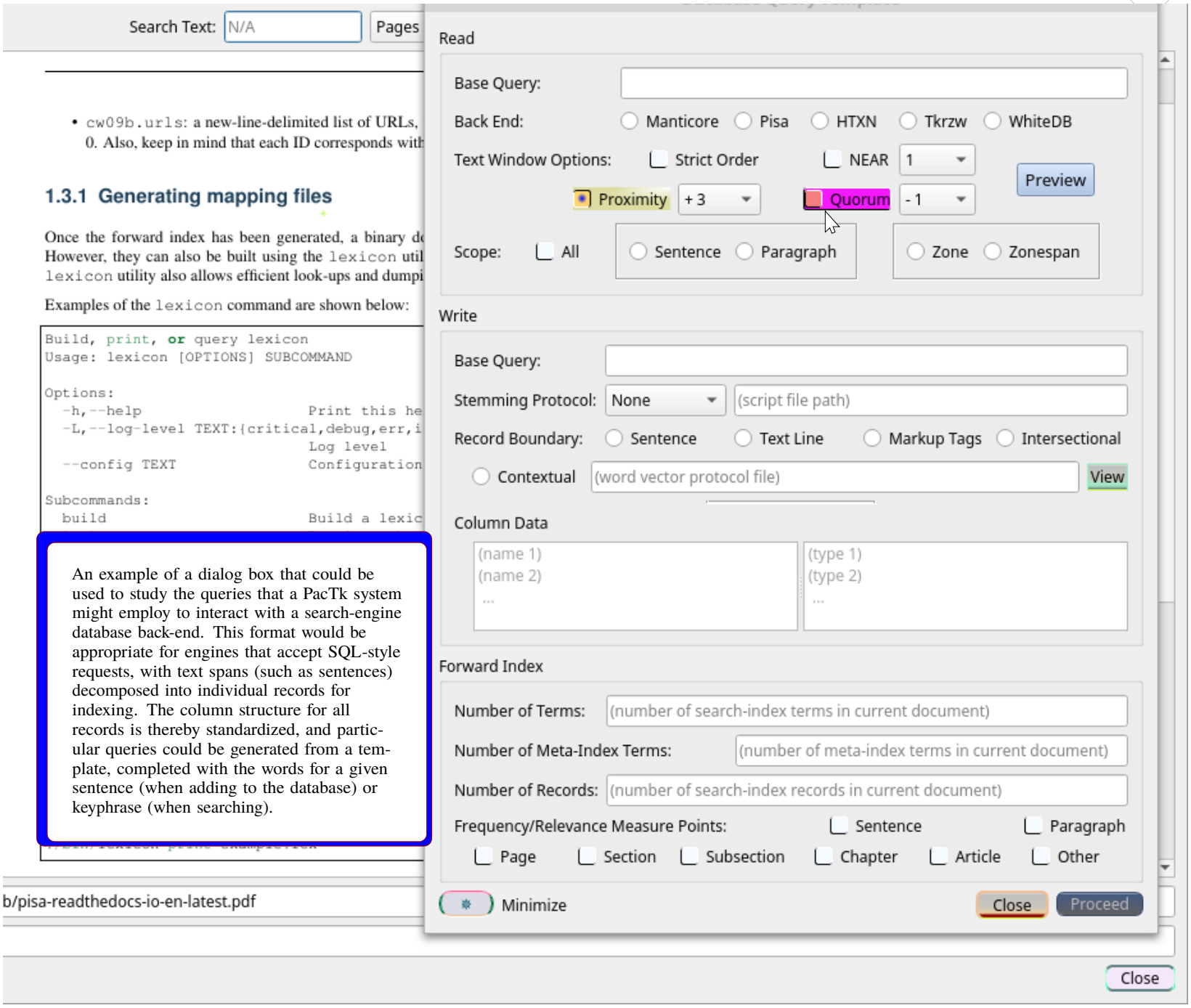



Figure 6: Configuring default back-end queries

**Accurate Search Portals** For users accessing search functionality through online HTML portals, extra programming is necessary to properly show results derived from PDF documents. Figure 14 (second to last page) shows two excerpts taken from publications included among one publisher's lists of subconcepts under the "engineering" heading. Both examples show irregular line spacing and improper rendering of text with distinct font-styles. It is clear that the intermediate files, from which PDF was produced as expected, were less successful in the context of HTML generation (an accurate PDF is displayed at the graphics bottom). These pages show a weakness in JATS-style encodings that can be alleviated by a sentence-data model where styling objects have both LaTeX and HTML output content.

Another source of transcription errors is character strings which are not ordinary language and therefore require special encoding rules. The book on the left in Figure 14[1] was a survey of biomedical software engineering in several domains, including text mining, radiomics, and immuno-oncology. Vis-à-vis text mining, the book reviewed CORD-19, a large open-access document repository curated by the Allen Institute for AI (AI2) to spearhead Coronavirus research. This corpus, unfortunately (and as acknowledged by AI2) had inconsistent (PDF-based) text extraction; e.g., in the aforementioned analysis the authors gave examples of a chemical formula that was printed in different styles in distinct publications reproduced within CORD-19, which could cause the overlap on that common semantic element to go unrecognized by text mining algorithms. In response to problems both with special-purpose and ordinary text representation, AI2 issued a "call to arms" encouraging publishers to adopt standardized machine-readable encodings as supplements to their PDF (and HTML) outputs: "Existing schemas like [JATS] can be too coarse-grained to capture all ... metadata elements, [and] representations ... vary greatly across publishers who use them. To improve metadata coherence across sources, the community must define and agree upon an appropriate standard of representation"[2] PacTk is an example of technology that could satisfy these goals.

[1] Amy Neustein and Nathaniel Christen, *Innovative Data Integration and Conceptual Space Modeling for COVID, Cancer, and Cardiac Care* (Elsevier, 2022).
[2] Lucy Lu Wang, *et al.*, "CORD-19: The COVID-19 Open Research Dataset," Allen Institute for AI, 2020. https://aclanthology.org/2020.nlpcovid19-acl.1.pdf

## Granular Locators

Current indexing tools are migrating to paragraph ids (rather than page numbers) for index preparation, although numeric locators still appear in the human-visible digital (and, of course, printed) versions. For best accuracy, it is worthwhile to partition paragraphs that contain block quotes, enumerations, or other subdivisions and assign unique ids to each part (see Figure 7). For example, segments of text before and after a block quote (plus the quote itself) can have distinct ids, even if they are each part of the same larger, encompassing paragraph.

Given this precision, digital indexes even in **PDF** documents can benefit from paragraph ids. The right-hand part of Figure 7 shows an index with color-coding to indicate such details as subdivisions (e.g., block quotes) and paragraphs split between two pages. In the depicted document, all locators are clickable and scroll to a precise location (top of paragraph or of page).

## Bidirectional Indexes and Text Extraction

Assuming index entries are pooled into an archival data structure, each entry is a potential cross-reference point between multiple documents. The concept of a "locator" may thereby extend beyond a single publication; from any one digital index users could initiate searches for matching entries in other books or articles.

To support such functionality, text encoding should describe structured objects (such as the date and string-name tags in **JATS**) wherever segments follow canonical notations beyond just natural language (e.g., citations, references, bibliographic entries, and quoting/transcription conventions). Also, meta-indexes via **PacTk** can enforce consistent encoding for special characters (e.g., quote marks) and structured segments (e.g., proper-name initials) in cases where different style guides have inconsistent recommendations (for example, whether or not to include spaces between consecutive initials). Treating such portions of a document as unstructured text content can obscure cross-references and potential micro-citations, as well as cause errors within manuscripts internally.

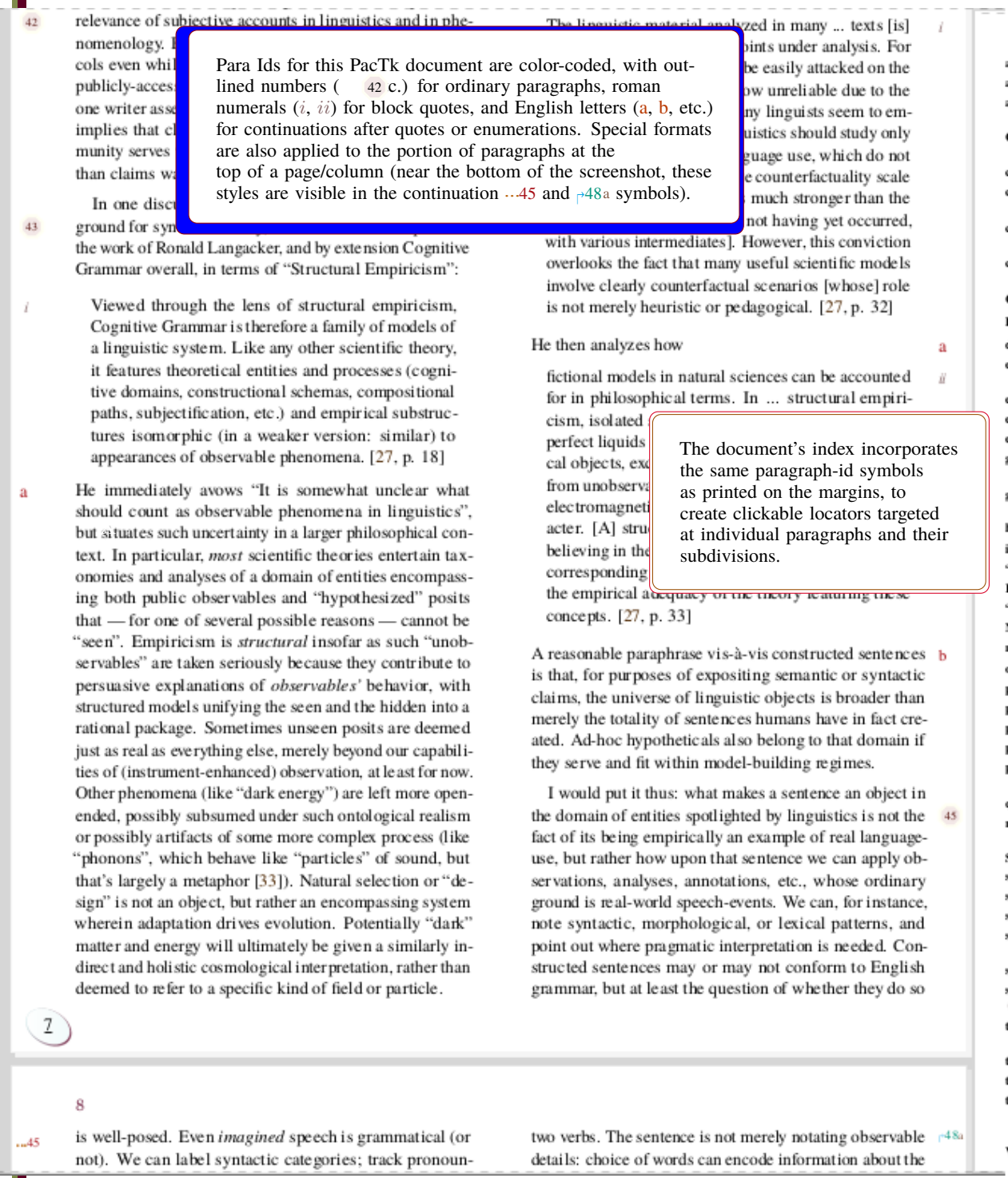



Figure 7: Subdivision formats for Paragraph IDs

### Indexing Digital/Multimedia Repositories

For some initiatives, authors deposit content in a variety of media, including but not limited to text documents. For example, the Cancer Biomedical Informatics Grid (caBIG) project is an oncology research network across multiple continents; institutions can join the network if they adhere to "Vocabularies and Common Data Elements" (VCDE) standards applying to APIs, metadata, provenance declarations, execution models, and so on, to support cross-network interoperability (for example, data from one institution could be plugged in to simulations from another). Preparing indexes for publications deposited to a structured research network such as caBIG (now succeeded by NCIP, the National Cancer Informatics Project) optimally includes aligning each index with semantic protocols. For example, index entries can be linked to VCDE terms wherever possible. GUI indexing can likewise address problems where users struggle to enact desired functionality because they are seeing outdated submenu instructions. Or, text- and index-search APIs may be subsumed under a grid's API policies; the shared versioning system could be applied to iterations of a manuscript; and metadata might cross-reference text publications with other submissions by the same (or affiliated) authors.

Similarly, according to standards such as FAIRdata, publishing findable, accessible, interoperable, and reusable data sets goes beyond just making raw data available on an open-access platform. For instance, FAIRdata defines documentation and metadata protocols, so that each statistical parameter (for example) is described with range, measurement, distribution, and similar analytic information.

On a scientific grid such as caBIG, many assets are executable code for "*in silico*" simulations. From a user's perspective, however, such executables often function as multimedia resources, producing two- or three-dimensional graphics that are pedagogically analogous to image series or videos. Such computer code thereby connects to publications with a similar rationale as videos, digital maps, and other highly-visual content types. As an example, analysis of habitat regrowth after a forest fire can be based on footage of actual locations *or* on agent-oriented simulations that provide mathematical models for stages of regeneration (which plant and animal species become reestablished at which point in time). Indexing and cross-referencing policies that are adapted for image and video material can thereby be extended to code libraries as well. Controlled vocabularies and other semantic standards apply to the familiar elements of computer code — procedures, classes, data types, inheritance/containment hierarchies, pre- and post-conditions, etc. — but in a "research grid" context graphical displays often also function as assets in their own right and can be annotated and indexed as such. A 3D simulation, for instance, functions essentially as a video whose specific frames evolve algorithmically as a product of parameters and initial conditions that may be modified for each run (indeed, videos could be taken directly from dynamically evolving GUI windows; see Figure 13).

Disciplines such as systems biology, environmental science, quantum cosmology, and neurophysics are characterized by research networks that explore complex systems from multiple angles, and leverage resources combining data, code, graphics, simulations, and natural-language text. PacTk can help authors expand indexing protocols to encompass this wider range of scientific content.

In particular, we propose a "Science Grid Node" protocol (Sign-index) to standardize code annotations according to FAIRdata principles, with an emphasis on indexing GUI logistics ("action-maps") and source-declared contracts. This protocol would reflect requirements for grid-computing architectures, with a unified meta-index paradigm encompassing data, code, and visual assets for FAIR-aligned research networks. The Sign-index model addresses concerns specific to modular software engineering, such as merging component-local with shared-database updates; discovery, validation, and integration of new modules; and reading intracomponent annotations for shared, searchable documentation (form fields, user-initiated operations, authentication, component capabilities, etc.).

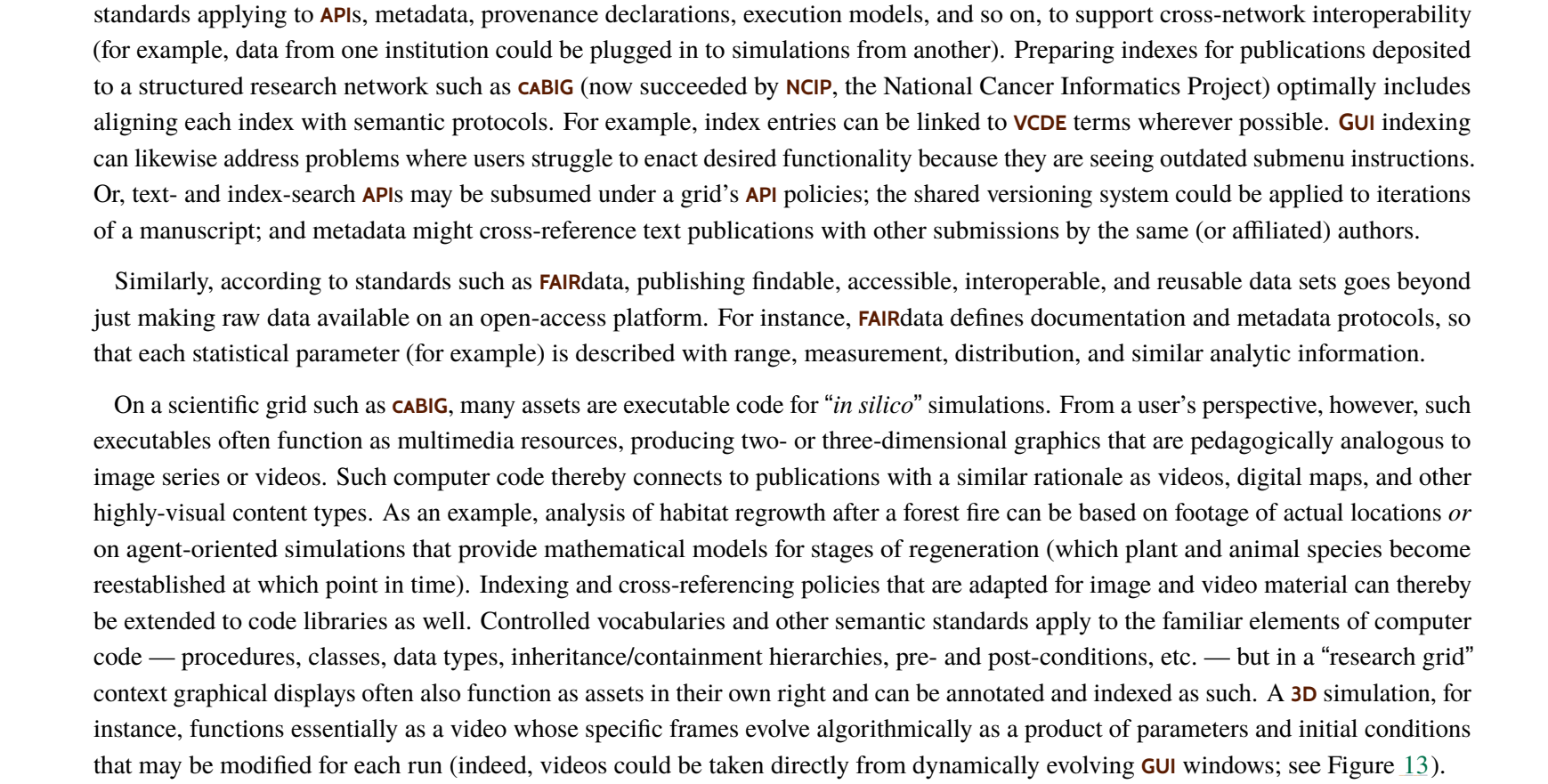



Figure 8: Index Search in a PDF Front-End Context

## PacTk in Comparison to Other Formats

Here are several points of contrast between **PacTk** and other text-representation formats that are currently popular. **PacTk** does not necessarily compete with these alternatives, because **TSOM** data can be converted to other representations as needed. However, it is useful to describe the strengths and weaknesses of some alternatives as a point of contrast.

**JATS**: The "Journal Archiving and Interchange Tag Suite" is currently the format that is closest to a *de facto* standard for academic publishing (**BITS**, or Book Interchange Tag Suite, is an extension of **JATS** to cover full-length books rather than individual articles). **JATS** is an **XML** format with over 300 tags having standardized semantics and inter-relationships. Many of these are for metadata related to publications' provenance, authors, dates of publication, and so forth, but some tags also define textual content in a structured manner. Because **JATS** is readily customizable, different publishers can add on their own extensions and tools to the basic tag set. As a result, there are often inconsistencies in how textual content is represented by different publishers even if they are both using **JATS** overall. Moreover, some publishers employ their own private (and/or legacy) **XML** formats for internal processing but map their files to **JATS** or **BITS** for sharing with other organizations. In totality, then, publishers' adopting **JATS** does not automatically alleviate typographic or content anomalies which can delay efficient document preparation. **JATS** by itself does not spare authors the work of carefully proofing intermediate manuscript versions.

At the same time, although there exists a **JATS** "Article Authoring Model" for those writers who prefer to submit **JATS** manuscripts directly, in most cases documents are delivered to publishers in formats such as Microsoft Word or LaTeX and must then be converted to **JATS** by copy-editors or others working internally on the publisher's side. This process cannot be fully automated, and many kinds of errors may be introduced at this stage. For example, numerous character sequences have to be modeled in **XML** via character entities or specialized tags, which among input sources may instead be present as Word symbols or LaTeX macros. Unless manually corrected, these transcription errors can produce subtle flaws that degrade the quality of final publications, due to inconsistent spacing (around quotes, parentheses, ellipses, long dashes, etc.), glyphs (e.g., straight versus angled quote delimiters and apostrophes), citations, and other text-discourse conventions.

One difference between **PacTk** and **JATS** is that for **PacTk** all textual elements are **C++** objects, rather than tags or character entities. Another important contrast is that **TSOM** focuses on representations at the sentence scale, whereas most **XML** encodings are oriented to paragraphs instead. In **PacTk**, such materials as quotes, footnotes, citations, discipline-specific symbols, and content set off via parentheses or em-dashes, are treated as objects either nested inside sentences or as containers holding one or more sentences in sequence (in some cases both). Discursive structures such as citations, quotations (with ellipsis and emendation protocols), and terms or proper names matching index headings, are modeled as class-instances algorithmically isolated from surrounding text, more tractably than **XML** conventions via tags or character entities. This forestalls many transcription errors that result from treating objects or delimiters with special functional roles (in quote, citation, or formula-notation contexts) as ordinary symbols or characters (such that **PacTk** is more computationally reliable).

**BioC** and **TEI**: these are two **XML** languages that are more granular than **JATS**/**BITS** in that they support sentence-level tags (although, as mentioned on page 1, such tags are optional and might be excluded from any encoded documents). Although **TEI** originated in response to requirements for humanities scholarship (e.g., preservation of historical documents) it has more recently been adopted in the natural sciences for tasks such as text mining (as in the **ISTEX**, or "Service infrastructure for text mining," project); **BioC**, meanwhile, is predominantly focused on biology and medicine. Both **BioC** and **TEI** only partially model the discursive constructions (quotes, footnotes, citations, etc.) that are the source of many transcription errors just mentioned, and also hallmarks of well-structured academic writing. As a result, they have limited value as internal representations for scientific manuscripts, although they serve well as interchange formats for environments such as PubMed. As pure-**XML** solutions, **BioC** and **TEI** need to be parsed into live objects of languages such as **C++** in order for detailed processing to be performed. For this reason, these formats are not direct alternatives to object-based representations such as **TSOM**.

**JSON**: Text-mining corpora frquently utilize **JSON** (JavaScript Object Notation) formats such as **S2ORC** (from "Semantic Scholar Open Research Corpus") primarily to encode written documents for text-mining methods based on Natural Language Processing. Given this focus, typical **JSON** formats do not rigorously model the *discursive* (as opposed to purely linguistic) facets which are important to academic writing, such as quoting and citation protocols. This is one reason why **JSON** is not currently a feasible option for publishing workflows.

Also, as with **XML**, **JSON** is only computationally tractable when first converted into programmable (canonically, Object-Oriented) data structures. These will necessarily be much more complex topomorphically than **JSON**'s object/array meta-model; consequently, code mapping to and from **JSON** encodings will be more complex and hard-to-maintain than direct **TSOM** serialization.

**Markdown**: Dating to 2004, Markdown is a popular format for material that needs only basic styling, such as headers, italics, underline, boldface, code excerpts, hyperlinks, and bulleted/enumerated lists. Markdown is widely used in social media, collaborative editing (including "wikis"), code documentation, and technology-oriented websites such as GitHub and Stack Overflow.

In itself, Markdown does not have sufficient detail to be useful as a publisher's internal text-encoding language. On the other hand, for many documents the authors' own contributions need not necessarily take on presentation details outside Markdown's scope, so it is reasonable for publishers to accept Markdown as a submission format. Because social media and tech sites are widely used, many authors have some knowledge of Markdown and, by extension, are not locked in to **WYSIWYG** (What You See Is What You Get) word processors. Non-**WYSIWYG** input sources are, in general, less error-prone than alternatives such as Microsoft Word.

For these reasons, although offering many more features than conventional Markdown, **GTagML** is based on Markdown and designed to have a smooth learning curve for Markdown users.

**LaTeX**: Notwithstanding that XML-based technologies (including XSL-FO, which employs style-sheet transforms to convert formats such as JATS to final publications) are becoming increasingly popular for generating PDFs, LaTeX remains a powerful tool for this specific part of the workflow. It appears that LaTeX offers authors and editors more fine-grained control over the final appearance of a publication, with the ability to manually override anomalies such as widowed or orphaned words and lines; to support details such as margin-notes, shipout overlays, and the use of sophisticated graphics for non-font symbols; and in general to perfect manuscripts according to the author's vision. As such, PacTk supports LaTeX generation and arbitrary LaTeX code may be embedded in PacTk input sources (without affecting the extraction of TSOM-modeled language/discourse content proper). However, LaTeX is problematic for *internal* text representation. While some discursive elements may be represented as LaTeX commands or macros, extracting these from presentation or utility code is a non-trivial process.

## Examples of Analyses Requiring Sentence Objects

The following cases may help to clarify why markup formats — when they are used not just for the interchange of text or metadata, but as intrinsic components of publishing workflows — can be difficult to use for certain kinds of search and content-management requirements. These are situations where computing over text/discourse elements represented as objects in computer memory is more efficient than traversing DOM nodes and hierarchies. As such, a framework such as TSOM, where sentences and other discursive units are already converted to code objects, is more convenient than (for example) XML files where nodes must be mapped to objects as a preliminary step.

**Categorizing Inner Documents via Annotations**: You are working with a large manuscript assembled from many smaller documents. Their authors have contributed PDF comment boxes associated with specific sentences, paragraphs, or sections of their text, and have employed a "handle" syntax to mark themes relevant to these content units, according to a standardized vocabulary. Your goal is to scan each of the comment boxes so as to identify all inner documents which address specific themes. That is, you want to build a mapping from theme labels to lists of inner documents (or chapters, etc.) which could be pressed into service after splitting the larger manuscript into smaller parts; or utilized to assemble glossaries, enhanced ToC pages, web navigators, and so forth.

To complete this task it is necessary to execute procedures that can obtain the content of all PDF annotations, indexed by page number. One would then search each annotation-text for every characteristic handle that indicates relevance vis-à-vis individual themes, building up an std::map style data structure whose keys are theme objects/labels. Following the comment box page number, one could moreover identify the chapter or inner document including that page, to serve as a value set for the aforementioned map (depending on the set up, more granular locations such as paragraph ids may be available as well). On the basis of that map data structure, the algorithms could generate additional code (e.g., for a glossary) or manipulate secondary files: suppose, for instance, that one goal is to split the manuscript into one file per chapter, or per inner document. The algorithm would then traverse the map structure to build a second one whose values are file names (while creating those files as necessary). These coding requirements — working with object-based data structures and local filesystems — call for programming in full-fledged languages such as C++ rather than queries or XML transforms.

As a related use case, consider how comment boxes might be leveraged to construct a printed index (or also, secondarily, a search-engine/vector-database sort of index) with attention to rhetorical details. When the text of a manuscript provides a citation or names a person or theory, authors could be encouraged — either directly through input sources or via PDF annotations — to summarize positions taken vis-à-vis that subject matter (e.g., endorse, critique, or neutral/evaluative). When the sources or comments are analyzed to extract those notations, the printed index could be extended by incorporating that material into index entries. For example, instead of an undifferentiated heading locating "Keynesian economics," the index could produce subheadings such as "Keynesian economics, arguments for"; "Keynesian economics, arguments against"; "Keynesian economics, mathematical models"; "Keynesian economics, and macroeconomic theory" and etc.

**Extracting Linguistics Annotations**: You are working with a linguistic article that contains a list of sample sentences/passages, according to a common linguistics presentation style. These examples are intended to demonstrate specific claims (about parsing structure, conceptualization, etc.) or to serve as a basis for subsequent analysis. Such samples — which might be obtained from written corpora, audio records, or simply invented by the author to highlight specific linguistic topics — are commonly used in this discipline and typeset in formats that visually distinguish examples from primary text. Moreover, the samples are often printed with id numbers, contextual comments, and sometimes special symbols representing phonology, agglutination, word order, morphosyntactic patterns, syntactic categories, etc.

Because this mode of presentation is so common, researchers could benefit from a searchable repository of these examples, as a special kind of language corpus. That is, multiple linguistics papers may be scanned for listed samples and each of those examples extracted into a common format, to be included in a multi-publication database. Interested readers could then browse the samples to find articles that address specific syntactic or semantic topics, particularly if each sample is classified according to the linguistic subject they exemplify (or serve thereto as a basis for analysis). Moreover, it is not uncommon for the same case study to be analyzed by multiple linguists, potentially creating interesting contrasts in theories and paradigms. The whole process would be accelerated if authors employ a common format (e.g., distinct XML tags) to notate the examples; but the full extracting and database persistence logic would need specialized programming.

**Embedding a Publication Viewer in a Chemistry Application**: You are working with an article about biochemistry, and the text includes discussions of compounds identified either by a common name or a chemical formula. The authors and/or editors have been attentive to encode these special terms in a structured format (e.g., special XML tags) and have also added informative suggestions via PDF comments. Your goal is to find all chemical terms/formulae which can be associated with molecular-structure files in formats such as PDB (Protein Data

Bank), MOL, CML (Chemical Markup Language), or similar standards. Those files may then be opened via science applications dedicated to visualizing and analyzing chemical structures in three dimensions (showing positions of atoms, atomic groups/arrangements, bonding strength, sub- and inter-molecular forces, and so forth). A similar example might be a chemistry textbook, with custom classroom software.

For this kind of context, special programming is needed to extract textual data about chemical names and formulae, and potentially to search for molecular files via science APIs. Furthermore — in the hopes of best leveraging 3D visualization capabilities — the PDF viewer has to interoperate with science applications, either through a common message-passing protocol or by embedding the entire PDF code base itself as a plugin or extension. Ideally, one or more molecular visualization applications themselves can be modified to work directly with these special PDF components, because — once 3D files are loaded into the software — the specific chemical compound being studied, and its properties, could be cross-referenced to locations in the original research manuscript (or, analogously, the students' textbooks).

## ▷ PacTk in the Context of a "Humanities Grid"

Use cases for PacTk in the context of scientific grid computing will be discussed below, but one should observe that many of the benefits inuring to research grids for physical science have analogs in the social sciences and humanities as well. Here are several use-case examples:

**Research with Computational Requirements**: As with natural science, a primary rationale for Executable Research Objects is to package analytic code that yields or supports conclusions presented in a publication (or at least demonstrates programming methods, assumptions, and algorithms intrinsic to that analysis). Fellow researchers can then examine analyses in depth (and possibly seek to reproduce conclusions). Ideally, researchers could rerun essentially the exact same code, perhaps against new data sets or tweaking parameters to examine the robustness/flexibility of theoretical models. Data transparency is also served, however, when researchers obtain simplified versions of the originary code for assessment and teaching. Dissemination *includes* but is not *restricted to* replication.

There are many branches of economics, sociology, law, political science, linguistics, etc., that work with empirical data sets and are therefore subject to FAIRdata standards, so that raw data should be published alongside analytic code and (as applicable) statistical graphics. To the degree that communities of scholars across multiple institutions jointly employ common protocols and link their research into searchable repositories, the end result becomes in effect a "humanities grid." In the case of linguistics, computational methods apply both to Natural Language Processing (e.g., LAPPS — the "Language Applications Grid" — and Europe's CLARIN infrastructure) and corpora curation/maintenance; in both cases textual content is represented in a structured manner suitable for annotation metadata to track parts of speech, dependency-grammar relations, lexical morphology, prosody, vocal inflection, and enunciation stress patterns/meter.

**Document Collections**: Preserving historical documents is important in many fields, either because the documents are themselves objects of study or as assets to scholarship (e.g., the collected papers of an influential writer). Archiving is more complex in this context than vis-à-vis contemporary publications because (for one thing) documents themselves — as physical objects — are often tracked and curated similarly to artistic or archaeological artifacts. Also, digital twins to historic papers — whether as images, transcribed to text, or both — typically demand more detailed markup than manuscripts prepared as digital resources to begin with (for instance, transcription ambiguities are recorded, or scribbled marginal notes notated alongside the corresponding text). This is a primary rationale for XML formats like TEI (which was originally based on the more expressive SGML standard) and concurrent-markup languages like TAGML, both of which may be subsumed into TSOM objects. Corpora of linguistic/acoustic samples (recent or historic) have similar requirements.

**Interactive Graphics**: Numerous use cases in humanities scholarship involve interactive GUIs displaying 2D or 3D graphics or multimedia. Analogous to historical documents, old photographs or videos often need analysis and preservation, a process that frequently includes picture or video annotations. Actions such as showing/hiding annotations, extracting video frames, navigating within image series, or invoking image-processing algorithms, all require GUIs that not only show the relevant media content but also have interactive capabilities. Moreover, image analysis sometimes comes into play in fields within cognitive science/phenomenology. For example, Computer Vision algorithms may emulate or model the biological visual acuity of humans or animals. In these cases, presentation tools can be especially useful if they illustrate Computer Vision workflows (perhaps showing individual stages in the transformation of pixel-data) and then explain how such processes might functionally or neurologically model phenomena within eyes and brains/minds.

**Translational Science**: Concerted efforts to "translate" scientific research into interventions or policies that benefit society — medicine, public health, environmental wellness, engineering, etc. — usually must recognize cultural and community factors alongside purely scientific ones. Public health collaborations, for instance, often depend on sociocultural and language experts to work directly with local communities. As translational science becomes more systematized, we can expect research teams/labs to increasingly include one or more individuals with a humanistic (in place of or in addition to a scientific) background, focusing specifically on social impact.

Meanwhile, data modeling, meta-science, and document preparation arguably solicit modes of theory and technique that are typical of humanities as much as natural science (the former tends to be more generalizing and philosophical; while "hard" science frequently hones in on narrow research questions/experiments). Written papers are still a major avenue for disseminating research, and herein science becomes absorbed into discursive and rhetorical spaces primarily modeled in the humanities. For instance, the book referenced on page 10 weaves together topics of law, ethnomethodology, and popular media; it endorses certain theories for family courts and critiques others. These rhetorical patterns could, in principle, be incorporated into document-view GUIs as interactive features, helping researchers find those parts of the book that most interest them.

The **Scign**-index protocol mentioned on page 10 is particularly aligned with "Executable" Research Objects (alternatively termed "Research Compendia"), yielding a proposed component-oriented model that **PacTk** calls "**ScignTier**" (Scienge Grid Node/Text-Integrated Executable Research Objects). In component-oriented programming, "microcomponents" are visually and functionally isolated parts of a larger application, such as a single web page within a web site, or (in the desktop context) a single application window. Modular software can be designed as an aggregate of many microcomponents. When reading publications, for example, microcomponents could provide functionality to view specific kinds of dynamic graphics connected with a manuscript, or review raw data files in an associated data set. **PacTk**'s **ScignTier** model extends the aforementioned **Scign**-index protocol to research-grid nodes (*Executable Research Objects* in particular). In this form of shared data, publications and accompanying materials are bundled into a single downloadable application — ideally as a mostly self-contained source file package, compiling to a binary executable that serves as an entry-point for both text and data.

To illustrate how microcomponent programming affects indexing and document-preparation, consider the screenshot reproduced in Figure 9, left (taken from the Open Hospital user manual). There are numerous areas within the **GUI** window with selections/options that may be connected to controlled vocabularies, including the list of checkbox labels for test results, and drop-down (combobox) values for "material" (there is no way to find a list of possible entries for this field without running the software or examining the code directly). There are also some varied actions users can perform through this component, including finding a patient id via their personal info; selecting dates from a calendar display; and recording exam results to a hospital database. Although these logistic and semantic details may be implicit in code documentation or user instructions, **ScignTier** would employ code annotations to specify this metadata in a machine-readable fashion; the entire software base can then have a single indexing and search entry-point (e.g., a query for the term "urinanalysis" should match this microcomponent as one option for the exam-type field). In addition, new microcomponents could be added to the hospital software systematically. Consider, for instance, that a more specialized exam report is needed for a particular clinical trial. Via microcomponent programming, that special-purpose page/dialog box could be integrated, and accessed by users, similarly to this generic exam window.

In the context of a research grid such as **caBIG**, content from disparate institutions are designed to interoperate. For example, one resource might be a Geographic Information Systems (**GIS**) data set tracking carcinogens in air or ground pollution plus incidents of different types of cancer; another resource might be software specifically designed to analyze and display this data (on a digital map). Such **GIS** attribute layers have a domain-specific nature that is difficult to accommodate with generic software such as **ArcGIS**. In particular, the **GIS** data points will represent either incidents (or measurements) of carcinogenic contamination or cancer diagnoses (organized by tumor classification). Apart from being governed by controlled vocabularies, such data points would have distinct **GUI** requirements (Figure 15 is an example). For instance, cancer varieties can be a terminological bridge for multiple data sets and leveraged via search **API**s. Lists of known carcinogens can likewise be incorporated into software that works with repositories curated by the **EPA** and other organizations. In general, context/menubar options, check/radio buttons, comboboxes, and clickable elements may all be indexed according to grid schemas.

For publishing and indexing, apart from semantic metadata such as standard nomenclature and "Common Data Elements" (defined by **NCIP**, for example), some **GUI** components (as discussed above) may function as graphics displays properly indexed as multimedia content. On the right in Figure 9 is a screenshot from the Cancer Imaging Phenomics Toolkit which has multiple data fields suitable for proposed **ScignTier** indexing, including textural segmentation and feature-extraction algorithms. In addition, the bioimage displays together with region-of-interest markings may function as "nanopublication" resources on a research grid, particularly with case-study examples used to demonstrate analytic methods (as with this picture). Illustrations appearing in publications may thereby be linked to microcomponents from which the graphics were obtained.

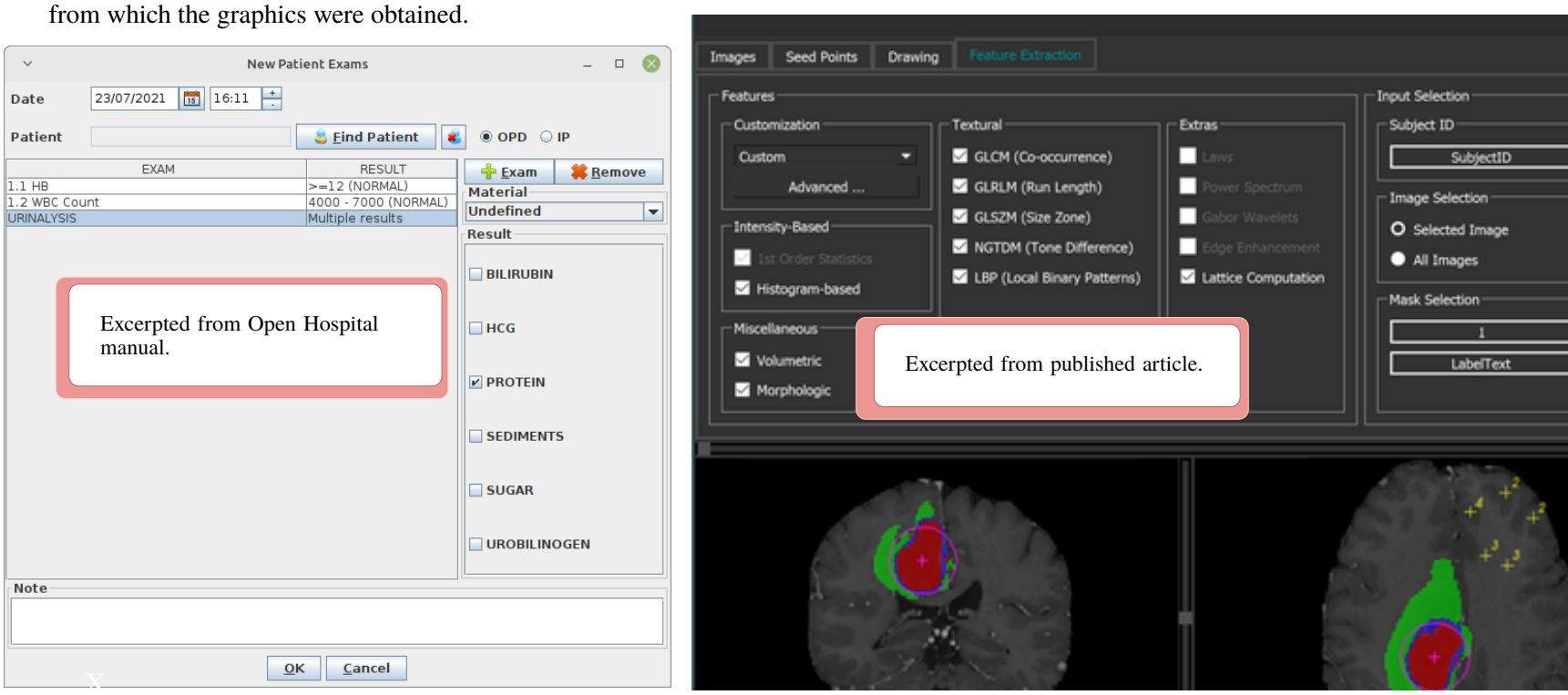



Figure 9: Open Hospital form (left) and cancer imaging window (right) as microcomponent examples

# Grid Architecture According to FAIR Principles

Grid Computing is an important part of contemporary scientific research. In general, a "grid" — in contrast to just a data-sharing network — combines data, code, and multimedia content. More specifically, a grid is designed with the assumption that users will execute software packages which are accessible via the grid, and that their primary reason for doing so is participating in large, decentralized computations, or because the software encapsulates novel research: quantitative or qualitative (e.g., agent-based) simulations, "in silico" experiments, analysis of new empirical data, algorithmically generated graphics, and so forth. In some instances, code packages may be downloaded and executed privately, e.g., on a scientist's own computer. In other cases, the code specifically requires special hardware (e.g., supercomputing/massive parallelization) and a grid enables researchers to run programs remotely, on software-hosting nodes that have the requisite capabilities.

It would be a mistake, however, to assume that grid computing necessarily involves technologies that require atypical execution environments. Instead, grid principles can overlap with FAIR priorities, because openly sharing simulations and analyses is one dimension of research transparency; and replication is a form of decentralization. Given a simulation with specific initial parameters, for example, any interested scientist should be able to execute the code themselves and achieve the same results (or, if probabilistic/stochastic methods are involved, at least *similar* results). Moreover, variations in initial conditions should affect simulation output in predictable (or at least explainable) ways.

There is also a pedagogical value in running and examining simulation and/or analytic code. Academic papers often present research methods and theoretical models in abstract, mathematical terms; observing visuals and experimenting with code parameters can help make the underlying research more understandable. This inures to the benefit of scientists who wish to assess (and perhaps reproduce) published research, as well as students who might use specific experiments or investigations as tools to learn about their ambient disciplines.

With this in mind, we can formulate several core principles of "FAIR" grid computing, including: • code packages hosted on a grid should be designed with as few mandatory external dependencies as possible; • where a given package has software or hardware requirements that are difficult to satisfy, a simplified version should be made available that would at least allow some provisional learning and evaluation of published results; • every effort should be made to ensure that the software executes in a usable fashion even in the absence of dependencies or preconditions that held true for those developing the code; • grid collaborations should offer replication-focused coding tools; and • there should be a clear set of actions, with as few steps as possible, allowing someone newly acquainted with a grid-hosted package — in particular, someone who learns about the code from a technical publication — to obtain and execute the relevant code. Researchers, especially, should be able to build and use the software in parallel with figures and text discussions featured in document manuscripts.

The following are several examples of how a properly FAIR grid might work, which are paired with "counter-examples" that discuss lacunae in *existing* implementations of research networks.

**Computer Simulations and Videos** A publication presents a new "*in silico*" investigation, such as the simulation of interrelated biological processes on a cellular or tissue scale. (Assume that the paper includes frames from a simulation video, and that a "supplemental materials" section links to the video in its entirety). The video itself records the output of a particular run of the simulation, with a given set of initial conditions. When reaching that point in the manuscript, the first thing someone might want to do, naturally, is watch the video. It is entirely possible for PDF viewers to include video-playback capabilities, even if few currently do. A FAIR grid could well include a PDF application with requisite multimedia functionality. Meanwhile, interested readers could also wish to examine the computer code which generated that video. Specifically, the video itself is merely a recording of simulation output (hopefully unedited, for data-transparency reasons). It should be relatively easy for users to examine *both* the video and the *in silico* run that produces it.

Counterexample The graphic pictured in Figure 13 is taken from a Computational Biology journal; the relevant article introduces CHASTE (Cancer, Heart And Soft Tissue Environment), a simulation framework written in C++. Switching from the PDF view to the relevant code and video is a tedious process: readers need to scroll to a "supplemental materials" section to download the **.avi** files, and manually find and launch the movies through a media player (this has no bearing on CHASTE as an important research tool; interop with text documents is as much the responsibility of publishers — PLOS ONE in this case — but in any event the process is needlessly burdensome).

More substantially, the process of actually identifying those components responsible for the video's content is convoluted, even after a reader obtains the relevant source code. The CHASTE package generates video files circuitously, and there is no straightforward documentation of what steps someone might take to reproduce the published video from the specific source files that run the depicted simulation. Moreover, neither CHASTE itself nor the project-specific simulation package (from Figure 13) includes a GUI component whose explicit role is to display CHASTE simulations directly. As a consequence, actually generating a viewable simulation that mimics the video requires detailed examination of the CHASTE source code and filesystem layout. Ideally, instead, that functionality would be localized to a single C++ class.

A Microcomponents Alternative According to FAIR standards, the process of switching between reading *about* the simulation in a PDF publication and *viewing* the simulation should occur as seamlessly as possible. One obvious step in this direction is to provision the PDF viewer with a video player. For example, the player could be launched by selecting such an action from a context menu activated with the cursor over the graphic showing one of the video's frames.

Once the actual source code is downloaded, FAIR principles suggest that (1) there should be instructions directly addressing how to find and run the code specifically associated with the referenced video; and (2) there should be a single GUI class and component whose specific purpose is to display (visuals of) simulations of the relevant kind.

More precisely, notice that there are several interrelated assets which could all be grid-hosted and indexed: the video; the GUI class (plus relevant .cpp files) implementing the window which actually shows the graphics it captures, both the overall window (presumably with various navigation and configuration controls) and the specific "widget" whose display directly yields the video feed; and the C++ code for the simulation's logic (in Figure 13's example, this code performs calculations like iteration-step computations and preparing values for a differential equation solver). There may also be a separate GUI window specifically dedicated to showing the simulation's initial parameters.

Such interrelationships should be noted on a multimedia grid — if the video file itself is a nanopublication, independently indexable and citeable, certainly the same applies to the GUI window which it records. But these connections are more substantial than just grid metadata. Interdependent components should have front-ends that are connected accordingly; for instance, context menus could allow users to navigate from the simulation display GUI to a video player and vice-versa. Similarly, context menus should launch a separate window showing initial values (as well as metadata for the current model, potentially) in both contexts.

In terms of search and indexing, all four components can be linked to any publication which discusses that specific video or includes illustrations derived therefrom. Terms specific to the computer code should be indexed in their source-file form (e.g., "DeltaNotchOdeSystem") alongside a natural-language equivalent ("Delta Notch ODE System") and related terms (*Notch signaling pathway*, *Ordinary Differential Equation*). Such semantics could be annotated on C++ files directly and then indexed when the overall package is deposited on a FAIR grid.

### GIS Attribute Layers Reflecting Multiple Scientific Disciplines

Digital maps are an obvious example of cross-disciplinary integration, because disparate data sets can share a common reference point vis-à-vis latitude and longitude coordinates. Public and environmental health is a case in point: such information as mortality rates and cancer diagnoses may be cross-referenced with data tracking environmental contamination (compiled either via testing air, soil, or water samples or by self-reporting from companies that use or transport hazardous materials). To visualize statistical correlations between these data categories, a good starting-point would be viewing all information together, superimposed on a GIS display. At the same time, rigorous examination of the underlying data should not depend on visual judgments, but can be confirmed by analyzing data sets computationally; for instance, quantitative methods could test whether there is an above-chance correspondence between particular environmental carcinogens and particular cancers. For such research, users need to navigate between map displays and digital "reports," or specialized windows for examining a cluster of interrelated data points: meter readings for detecting a particular class of chemicals at specific time intervals, say, or "toxic release" records with numerous geospatial, company, and location fields as well as cheminformatic details about individual accidental discharge incidents. Effective GIS components will merge both the geospatial and the records or "attribute" data in systematic ways.

#### Counterexamples

The preeminent commercial digital cartography software is ArcGIS, and many geospatial data sets are designed to be accessed specifically by this application. As a free (FAIR) alternative, authors also often create web-based map displays, where base maps and domain-specific data layers can be superimposed with only a few lines of JavaScript code.

The problem with web-based maps is that merging disparate GIS data layers is more difficult than creating a display for a single data set; and, also, navigation between browser maps and dataset record displays is harder to implement. On the desktop, modifying ArcGIS directly is infeasible. Open-source alternatives such as QGIS are available, but they are not well suited for domain-specific data profiles. For example, QGIS only allows users to examine GIS records when transformed to the columnar formats specific to digital maps, which are often much different than the information's internal structure. Web maps, lacking multi-window displays, tend to have a similar problem.

#### A Microcomponents Alternative

In lieu of an open-source GIS *application*, a FAIRdata grid should provide GIS *components* for reuse by individual publications. Here in the current document, Figures 11 and 15 are taken from ProPublica reporting about pollution (from 2019 and 2021, respectively). For the latter example, ProPublica provides maps centered at regions with (apparently) elevated environmental-health risks, but epidemiological and contamination data is not directly included, although (as in the bottom portion of Figure 15) researchers can locate related data sets elsewhere. In an academic publication, authors might wish to build an integrated map with attribute layers reflecting *both* contamination (toxic release; carcinogen) and public health (cancer diagnoses; mortality rates) data sets.

For publications on a FAIR research grid, the grid itself could provide a skeletal GIS microcomponent that authors may complete with their project-specific information. Such a component would feature a window showing base-map layers alongside GIS point- or region-locations using icons or coloration to signal data points or categories. The display could then be annotated as a multimedia resource, cross-referenced with associated publications, by analogy to how an iterative display of *in silico* simulations functions (similar to videos) as multimedia content as in the previous use-case. When implemented as a microcomponent, such a map display would have control over context menus and other features for navigating between GIS and data-record views. Moreover, the component could be embedded in software specific to the disciplines being integrated — for example, embedded simultaneously in a hospital consortium EMR system (suppose a clinical collaboration adopts a program to study correlations between public and environmental health) and in cheminformatics applications (e.g., to investigate biochemical properties of hazardous compounds through molecular visualization, atomic force metrics, biosimulations, etc.).

### A Modular EMR System

At the basis of any Clinical or Hospital Information System is a collection of EMR "forms" (in common usage, an electronic *medical* record is one primarily used internally in a single institution or medical practice, whereas an electronic *health* record — EHR — is more "longitudinal" and can share data about a single patient *between* care locations). The term "form" is

informal, and EMR displays are not exactly like web forms, for example, but the basic idea is the same: forms have a collection of labeled data fields, potentially alongside areas for graphics as well as navigation links to other forms. Typically forms have text-entry areas and/or drop-down selectors so that someone (with proper authentication) could modify as well as view the form's contents.

In this sense, GUI forms are not grid assets *per se*, but they have many similar qualities from a microcomponent point of view. Individual forms are largely autonomous: each one's contents are not intrinsically dependent on another's. In a real clinical setting, of course, information such as patients' personal details needs to be synchronized, so EMR instances are interdependent. Nonetheless, it is certainly possible to initialize forms with mockup data for development, testing, training, and pedagogical purposes.

Meanwhile, EMR technology does align with bonafide grid applications in some contexts. For example, clinical trial management software, as well as diagnostic technology, has patient and cohort reports structurally similar to EMR forms — and, if translationally the research yields concrete benefits, some version of those reports may indeed become EMR records *de facto*. Thus, FAIR grids might certainly host assets that functionally have an EMR profile. For instance, research on novel bioimaging techniques could provide a computational platform to annotate images with representations of Computer Vision feature vectors. Such research components would potentially migrate to a new class of radiomic reports — thereby integrated with EMR systems — under translational deployment.

**Counterexamples** Several open-source EMR and Hospital Information Systems libraries provide a good reference-point for analyzing software implementations. These components are especially important for patient care in "under-resourced" communities, where the option of selecting open-source environments (such as BioLinux) can eliminate licensing fees (a considerable savings in hospitals with a large staff, needing multiple installs per program). Among extant projects, FreeMedForms (C++) and Open Hospital (JAVA) are representative examples, embodying years of development efforts by large programming teams (an analogous project, HospitalRun, was discontinued in October 2025). All of these libraries are desktop-oriented and designed to operate despite intermittent (or nonexistent) internet access, which can benefit communities with spotty web connections, and also clinics operating under emergency-response conditions.

Taking FreeMedForms and Open Hospital (see the left part of Figure 9) as examples, one noteworthy issue is that of *internal* dependencies. Principles of FAIRdata address *external* dependencies, but internal-dependency complications should be minimized as well. For example, both FreeMedForms and Open Hospital are build around SQL databases. That's to be expected, but any JAVA or C++ developer would counsel that applications should not be *crashed* by faulty database connections (database configuration can be notoriously complicated). Unfortunately, neither FreeMedForms nor Open Hospital appear to be capable of operating around a missing SQL connection.

Another issue evident with FreeMedForms is an overreliance on plugin architecture. In principle, projects that concentrate much of their implementations within semi-autonomous plugins have proper, flexible modularization. However, designing *everything* as a plugin has the opposite effect, because the entire plugin architecture becomes an internal dependency upon which all components rest. A simple error in the build workflow, or rearranging folders from one version to the next, can make plugin-dependent code unusable.

**A Microcomponents Alternative** A FAIR grid approach to EMRs would recognize that forms have use-cases even without actual data. Simply previewing a form with no information at all — fields left blank or completed with default values — can help a user understand how the form is organized, and become familiar with the window's GUI controls and capabilities. For more substantial work, EMRs could be populated with fiat sample data for testing and documentation purposes. As a workaround for broken database connections, local filesystem-based persistence might be employed to track a few records for users who are first learning how to use the software.

The facets in which forms are *not* autonomous are those involving real-life patients, as well as interconnections between EMR varieties (from lab results to inpatient equipment readings, for example). Outside of these details, forms can be implemented as individual standalone programs. In some contexts, a user or researcher might be interested in just one specific type of EMR (e.g., if they wish to create a similar form for use during a clinical trial). In other cases, users could be offered at least a small collection of particularly important forms, which they could visit for tutorials and other exploratory goals (imagine the scenario where a HospitalRun administrator seeks to evaluate still-maintained alternatives; part of the comparison is applications' "look and feel," which is data-agnostic). Any EMR system should have a collection of basic controls that are "core" components built alongside the main non-GUI logic (*not* as plugins), and there should be one clearly identified primary application that is essentially a container for those core components, with users being able to select and visit them in turn. Third-party developers should find and study this application in an intuitive manner; in C++, say, there should be exactly one main.cpp file, or at least an obvious folder location where that file is understood to be a starting-point for viewing essential EMR types.

Where a plugin system *should* be adopted is to allow third-party developers to extend and fine-tune the system for an individual institution. The digital portions of projects such as clinical trials, community outreach (e.g., specialized maternity, vaccination, or mental-health projects), standardized treatment protocols, patient-centered approaches, etc., should be implemented through collections of special-purpose EMR forms which are coded using existing forms as reference examples. When properly tested and vetted, new EMR types should be introduced into the system using semantic annotations and interface-design conventions such that the basic usage patterns (context menus, cross-component navigation, user authentication, etc.) are consistent across all microcomponents. This may seem self-evident, but existing frameworks do not provide sufficient support for such modularization to occur absent concerted effort; for example, implementing context menus according to systemwide protocols requires supporting code more detailed than just the basic QMenu/QAction mechanisms from QT. In this sense, balancing component autonomy with systemwide protocols is a concern common to EMRs and to FAIR microcomponents.

Rethinking the relationship between pure and applied research — and between universities and communities — means a changing landscape for publishers as well. Although all biomedical research (and scientific work in general) is motivated in principle by practical benefits, translational emphases promote the institutional and informatics foundations for "bench to bedside" translation to happen as quickly and comprehensively as possible. Scholars in the field of "translational science," for example, have developed a five-part model of the evolution of practical interventions (also called the "translational continuum"; see for instance https://www.iths.org/investigators/definitions/translational-research) of which the fourth involves commonplace adoption of (once-)novel practices in clinical settings, and the fifth marks concerted efforts to adopt useful interventions in a wide variety of care environments (e.g., not restricted to protocols available only in a few hospitals). Such models suggest roadmaps to help guide deployment once research progresses through multiple stages of concrete implementation. Published materials (not necessarily restricted to books and journal articles) may reflect data and observations at each of these stages.

Apart from theoretical models along these lines, the key details of translational deployment are often institutional and socioeconomic in nature. Sustaining research projects sufficiently to rigorously assess empirical outcomes is a different process than procuring funding, labs, and teams to work through initial objectives on purely scientific grounds. Some innovative software initiatives (e.g., the caBIG Digital Model Repository) have been discontinued; other frameworks, such as the University of Pennsylvania's Cancer Phenomics Toolkit, evince novel biocomputational ideas but have not been scaled up or replicated beyond their academic point of origin. These suggest fortuitous connections between technical and translational priorities. Further development and adoption of promising software, interventions, or clinical protocols can be spurred by goals to make effective health care options available to a broader spectrum of the community. Research initially supported on purely technical grounds might then be further expanded by emphasizing social/translational benefits.

From a publishing standpoint, translational science implies that a research infrastructure should sustain and evolve over multiple phases of a scientific project, which extends beyond the time-frame of individual books or articles. This is reflected in part by protocols such as Executable Research Objects where "supplemental" materials (graphics, data sets, utility computer code, and so on) are standardized. It is also manifest in notions of replication and data transparency: models such as Minimum Information Standards that illustrate experimental/data-acquisition modalities to facilitate subsequent investigations confirming or extending the scope of published work.

More broadly, translational priorities imply that variegated stakeholders — not only scientists but also practitioners, funders, government representatives, etc. — are best served by integrated packages describing the scope, goals, and (possibly intermediate) results of research projects. In current practice, multiple files and resources associated with academic publications are often scattered in different places, without clear provenance or interoperation. Specifications such as the Research Object protocol only partially alleviate this problem.

Against this background, PacTk's proposed SignTier specification honors the goals of both Executable Research Objects (as a paradigm for dissemination of scientific works) and Translational Science (as a collation of best practices for emphasizing social benefits of research innovations). According to this protocol, individual publications exist in the context of a "Project Archive" that has a singular entry point both online (potentially integrated with publishers' search portals) and through downloadable resources/data sets. Materials associated with a given publication, aggregated through this entry point, would summarize translational possibilities for the documented research and hopefully provide a foundation for institutional deployment(s). Individual scientists or students could also utilize Project Archives as encapsulations introducing their interests, background, and expertise to fellow researchers as well as those whose responsibilities include assessing the merits and adaptation potential of projects originating in academic (as well as private/independent) environments.

## Data Publishing and Multimedia

Contemporary data transparency and FAIR sharing (Findable, Accessible, Interoperable, Reusable) standards introduce new challenges for publishers. In this environment, published documents such as PDFs serve as only one part of a package of files, alongside research data sets and, often, numerous forms of multimedia content (potentially including audio, video, image series, 3D graphics, and statistical/data-visualization tools and diagrams, plus domain-specific interactive formats). If desired, PDF viewers could themselves be implemented as one portion of a multimedia suite, with functional motifs duplicated across the various components (Figures 11 and 12).

As a "publishing accelerator" toolkit, PacTk has several components to help publishers construct these supplemental materials, including cross-referencing text documents with associated data/media assets, and ensuring FAIR usability. Here are several examples:

**Deserialization and Dataset Tools** Individual scientific disciplines are often associated with domain-specific data publishing formats that are paired with dedicated deserialization tools (i.e., computer code that reads raw data files into live memory, without relying on special-purpose software). Such code libraries are often maintained by individual institutions or organizations — which might be academic departments, research labs, or independent groups — on behalf of a broader research community. Examples include (among many others) FASTQ and VCF for genomics; HDF5 and ROOT for physics; FITS and NetCDF (astronomy and earth sciences); PDB, MOL, SBML, and CELLML (biochemistry); IFC and BIM (architecture/engineering); SEGY and BAG (geophysics and oceanography); CoNLL-U (computational linguistics); DICOM and OMETIFF (bioimaging); and GIS-indexed attribute layers and shapefiles for geospatial data sets.

Some domain-specific file formats are derived from general-purpose specifications such as XML, CSV, and JSON. Others employ

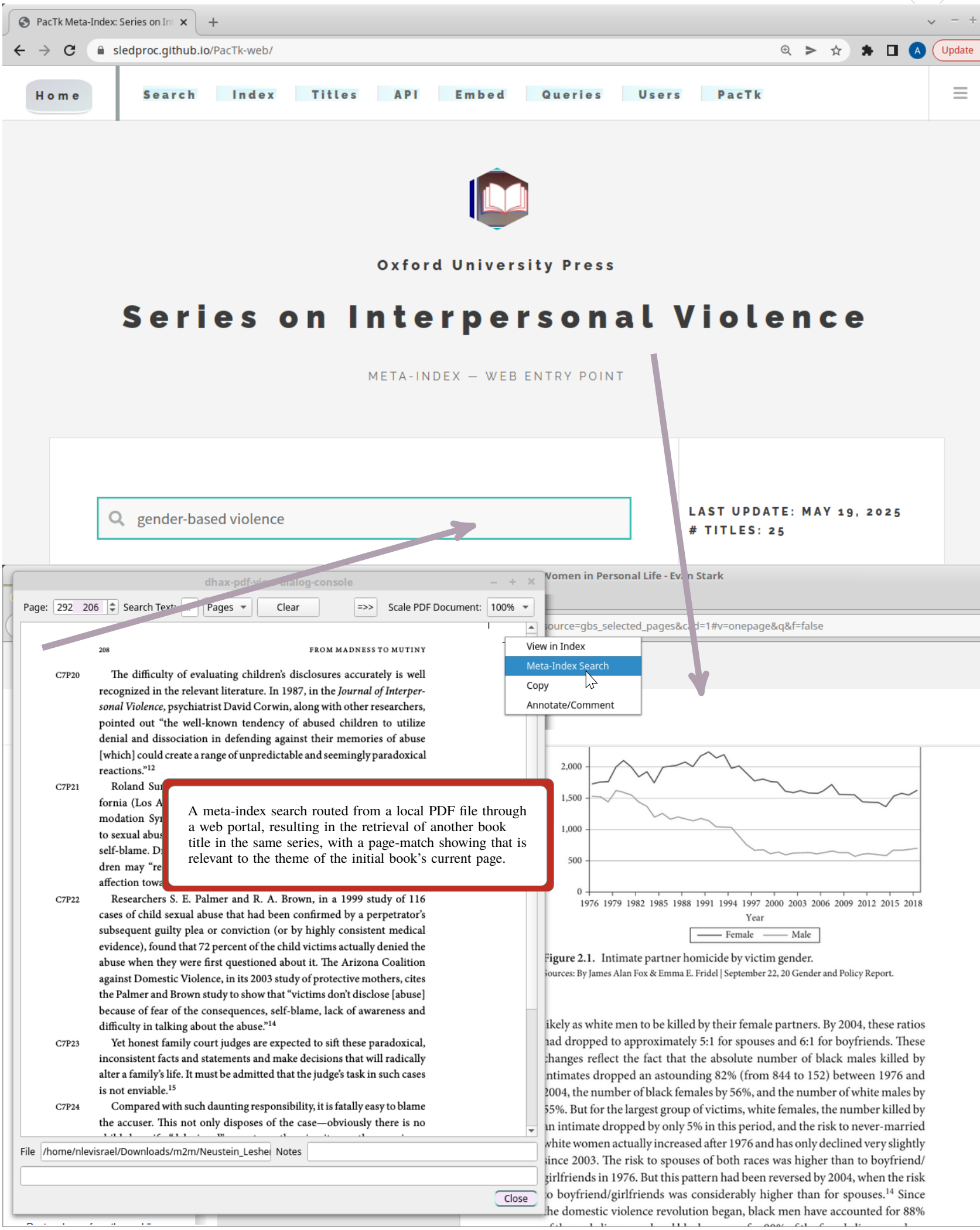



Figure 10: Meta-index search through a web portal

idiosyncratic text and/or binary serialization protocols, such that custom parsers are necessary so as to consume byte- or character-streams from relevant raw data files. Even when the actual encoding is performed via a generic standard like XML, dedicated deserialization

code will still be needed — ordinary parsers may convert a given XML document to a DOM infoset, but extraction algorithms still must navigate the document hierarchy and obtain text strings from element nodes, which become the input for initializing object data fields.

PacTk seeks to address these issues by offering a platform-independent Virtual Machine that is easy to host within scientific applications or also build as a plugin or standalone program. Scripting or query languages can then emit code proper to this VM to take responsibility for deserializing encoded data sets (whilst, as another use-case, the VM could be embedded in C++ classes/microcomponents as a runtime contract-verification tool). Depending on format, VM instructions might navigate through graph-like structures in a standardized profile (like XML) or implement and/or integrate with parsers for custom input file types, through postprocessors or rule-match callbacks.

Via these technologies, publishers may ensure that data sets are accessible so long as there are deserialization libraries targeting a PacTk VM, rather than duplicating effort (requiring numerous libraries for different operating systems and programming environments).

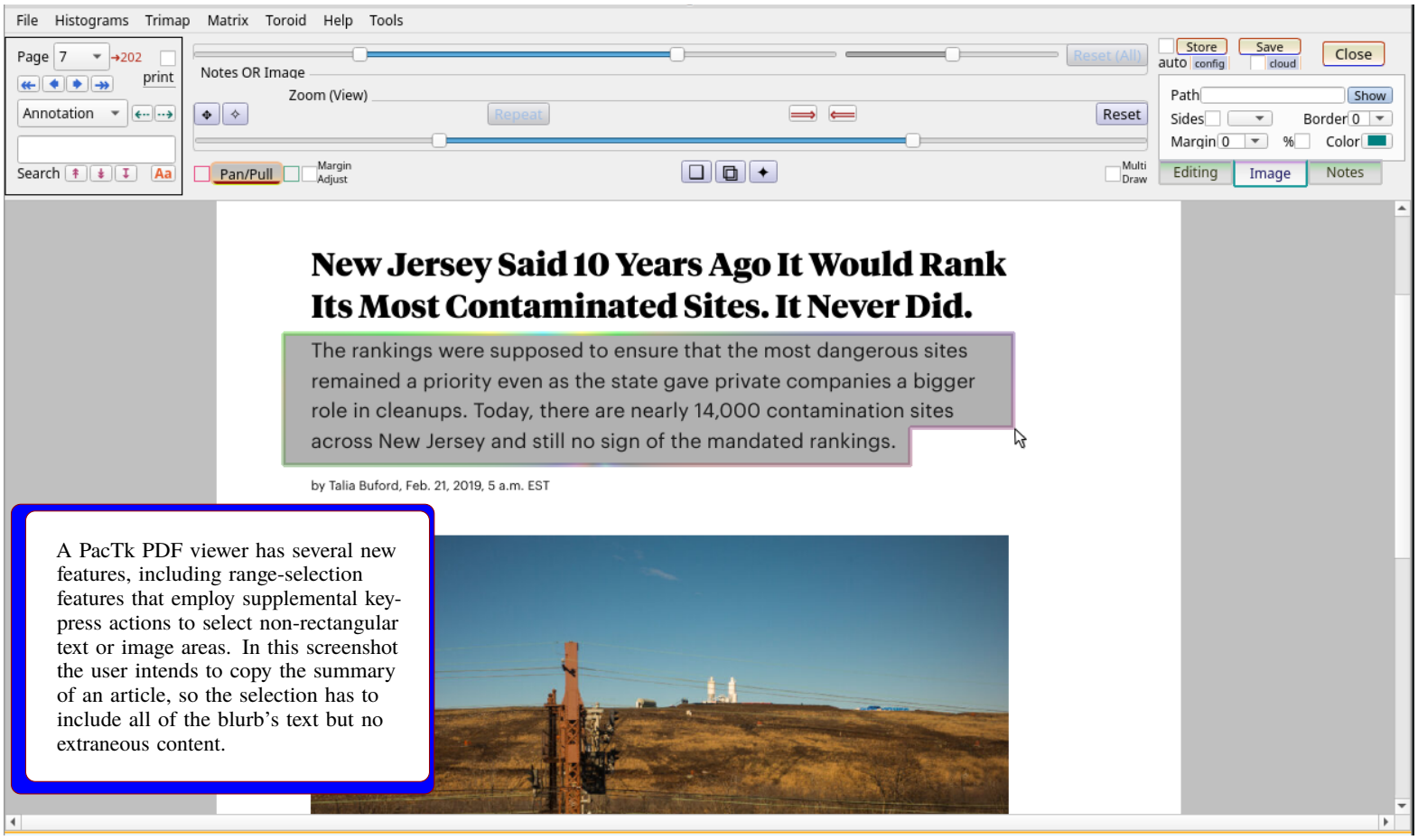



Figure 11: Flexible range-select features

## Analytic and Demonstrative Code

Many data sets are published alongside copies or samples of code used to analyze the raw data. Code sharing is important for transparency (allowing outsiders to double-check the accuracy of calculations and conclusions drawn therefrom) and reuse (replication of experiments/research projects). Data publications (bundled as Research Objects or similar scientific packages) may also include implementations for plugins targeted at applications charged with loading the raw data files. PacTk's object system includes several classes encapsulating plugin functionality (such as dynamic-library symbol resolution) and components exposed by host applications to extension modules (main-window taskbars, top/context menus, filesystem access, etc.).

Although such accompanying code might be implemented in many different languages and environments, the VM principles identified above for deserialization tasks also apply to code specific to research data sets. Here, too, accessibility implies that dataset code should be available in a cross-platform manner. Broadly-usable code libraries for (e.g.) deserialization obviate the need to achieve interoperability via reified/canonical model-representation standards alone. For example, Semantic Web nodes could be annotated with information about libraries that are able to unpack objects whose data fields supply values to nearby sites in the node's shape-constrained neighborhood. Moreover, implementations should proceed via conventions suitable for GUI wrappers and interop (in particular, microcomponent protocols described earlier) because research methods will often rely on domain-specific scientific or academic software. Ideally, users will examine data sets with the aid of plugins for applications familiar to the relevant research community, or at least custom-built software that can interoperate with those domain-specific technologies. Unfortunately, many of the most popular scripting formats in scientific/research contexts (e.g., Python, JavaScript, and R) have noticeable limitations vis-à-vis native-GUI integration.

PacTk enables a different approach, wherein languages can be customized to individual data sets and target pure-C++ VMs whose entire parsing, IR, and runtime stack is embeddable (as source files) in C++ GUI applications with no outside dependencies. Compilers and runtime engines for dataset code may thereby be released as part of the publication package itself. The VM and peer components could also be built directly into PDF viewers, multimedia applications (for videos, digital maps, etc.), and scientific computing platforms.

Another **PacTk** benefit for dataset code involves documentation and pedagogical features. Here, code serves not merely to derive specific computations but also to demonstrate the acquisition methods and analytic protocols relevant for a research project. Such details as units of measurement, expected ranges, typed coordinate systems (e.g., ensuring that procedures expecting horizontal-then-vertical coordinate pairs are never called with the converse), compile-time/statically-analyzable pre- and post-conditions, and so forth, help dataset code to serve as a resource clarifying scientific principles and theoretical models that drive published findings. The **PacTk VM** includes overload-by-contract, dependent typing (a version thereof compatible with template metaprogramming, as such with some technical differences from, say, **IDRIS**), native interop (with static-compiled as well as dynamically loaded libraries) and user-defined calling-convention features differing in some technical details from typical language interpreters, resulting in a scripting environment specifically built for open-access data sharing and deployment of data packages as pedagogical tools. Also, script syntax can be extended through a **VM**-like macro system supporting customization via novel grammar rules, parse-graph manipulation, and bytecode generators.

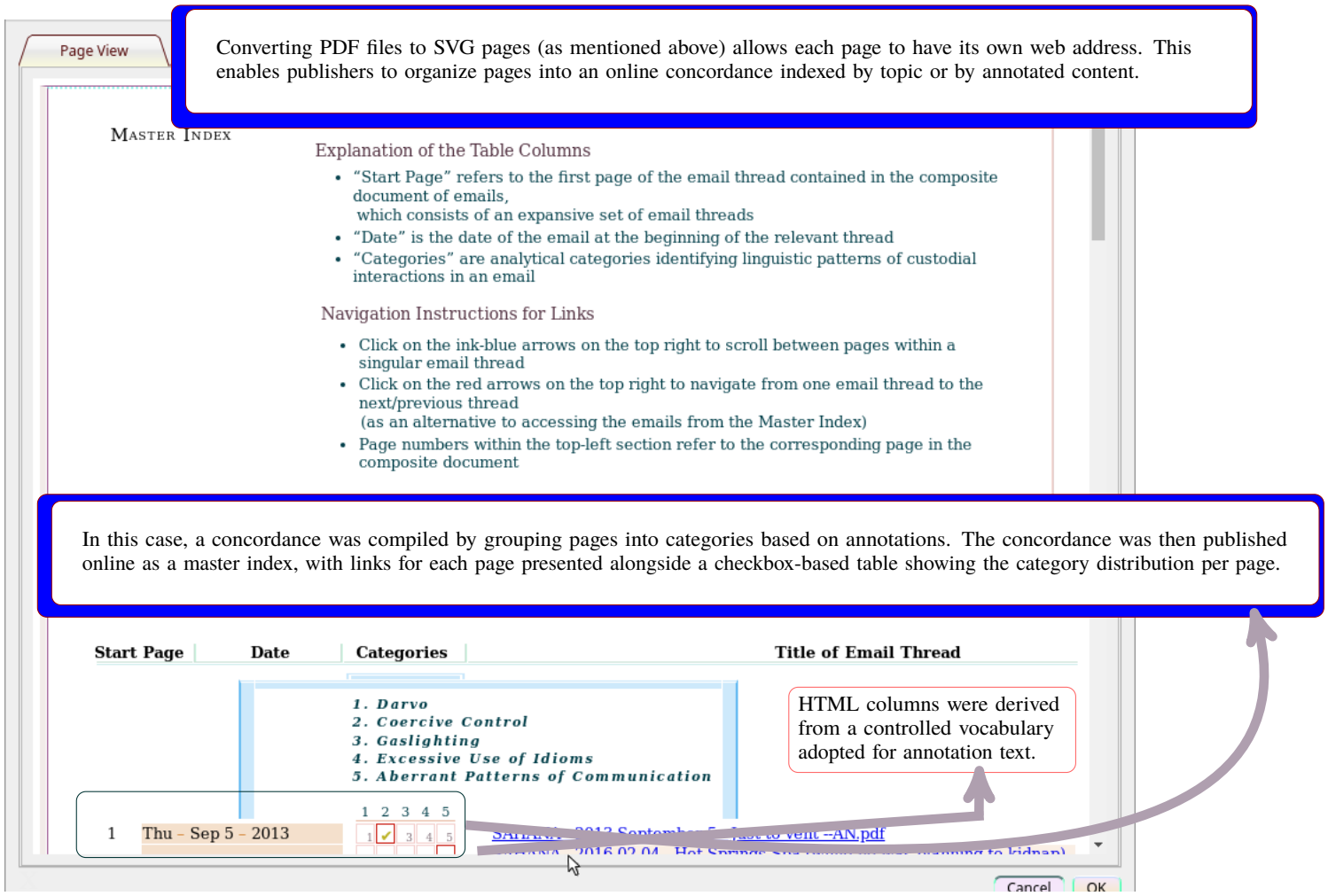



Figure 12: A category-based web-accessible index table for SVG pages

**Full-Text Query Languages** As discussed earlier in this document, **PacTk** can be utilized as a code library by calling methods on **C++** (specifically, **TSOM**) objects. At the same time, a **PacTk** Virtual Machine supplies an environment for compiling scripting or query languages. These functionalities could be merged, such that the process of extracting information from a **PDF** manuscript, meta-indexed repository, or bibliographic database might be achieved by formulating and evaluating part of the code in a specialized query language. The techniques involved would be similar to employing **XQUERY** or **XQSE** expressions to work with **JATS** or **BITS** files.

**PacTk** features its own flexible front-end parsing framework (mentioned above relative to **GTagML**) that could be used to program query grammars in an extensible, adaptable manner. By leveraging full regular expression capabilities, callbacks to arbitrary code (instead of generative production rules), and directly-executable grammar files (with no separate compilation steps), the **PacTk** parsers are more flexible than conventional **LEX/YACC**-style compilers. Meanwhile, the **VM** runtimes allow query evaluators to recognize basic scripting constructions such as locally-scoped variables and calls to arbitrary boolean-valued procedures as filter parameters.

In totality, such features permit text/bibliographic query systems to be implemented in a self-contained and cross-platform manner, with customizations specific to individual publishers and even to each particular journal or book series.

**Multimedia Cross-Referencing** When a document (book or article) is explicitly paired with a published data set, information about the supplemental package should be included in index/search metadata. This would enable keyword searches to be extended to raw data fields, as well as data-structure parameters (column labels, source-code annotations, procedure names, etc.). Moreover, sections of text may be isolated as cross-reference targets for data-structure parameters as well as individual objects/records. For example, a specific column or field in tabular data structures could be cross-referenced with the paragraph in article text where that field is discussed (its observation/acquisition methodology, units of measurement, valid range delineation, statistical distribution, and so forth).

Cross-referencing between research papers and data sets is a special case of multimedia cross-referencing, where annotations defined in the context of a specific media type assert semantic or thematic relations to contents of a different modality. Concrete examples of multimedia cross-referencing include geotagging video frames; linking CAD digital twins with product specs; annotating Regions of Interest within image series with coded labels associated with published raw data; information within natural/social science cross-disciplinary areas such as medical humanities, translational sciences, socio-agronomics, and civil engineering; embedding tags in 360° displays whose content includes URL strings that encode identifiers for dataset objects; developing color- and/or shape-coded attribute layers for GIS overlays (so users can browse between digital maps and structured object displays); and constructing numeric tuples from 3D graphics scenes such that particular camera orientation/location and zoom levels could be recorded as a unit, allowing relevant 3D content to be repeatedly visualized from an identical perspective (in other words, links are embedded in PDF files, web pages, software, or any other context so that users can reconstruct a specific 3D state, as constituted by camera angle/position and zoom settings).

The challenge when implementing multimedia cross-references is to ensure that viewers for two or more divergent media types may properly interoperate. With correctly interpreted cross-references, application users should be able to navigate from a window dedicated to one media/file type to a window rendering a different format, and correctly isolate the portion of a file (not just the overall file) implicated in a cross-reference. For example, geotagging video frames should enable users to pause a video at a particular location and then, via the geotag, switch to a map display centered on the geospatial coordinates visible in the video (consider videos documenting progress in an environmental remediation site; or infrastructural and socioeconomic implications of Zoning and Land Use ordinances). Conversely, a GIS attribute layer might represent locations for which video content is available, allowing users to watch the relevant videos starting from the appropriate time-point. Such interoperability is only possible if the components and/or applications responsible for rendering the two content-types share a common annotation and message-passing protocol. In the context of PacTk, this could be achieved via TSOM "message" objects that encapsulate remote-method requests and parameter serialization via TagML-like structures.

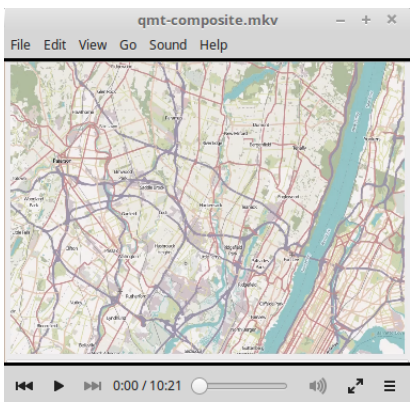
**Science and Humanities GUIs** Data publishing according to FAIR sharing standards often requires numerous project-specific resources in addition to raw data files. Fellow researchers typically need visualization and computational tools to fully leverage published data sets for evaluation and re-use. Unfortunately, relying on generic software to access published information packages is often suboptimal. For example, CSV files might be loaded into spreadsheets and/or statistical applications, but simply viewing data files in tabular fashion (even if the information is conducive to such structures) provides only limited value. The same is true for statistical plots and diagrams. Effective evaluation and re-use involves front-end programs that could provide both visual overviews (e.g., via charts, graphs, or histograms) and functionality for examining individual data objects in detail (e.g., via custom-designed GUI windows).

Furthermore, many data sets span a variety of structural profiles. For example, consider information tracking environmental pollutants, such as published by the Environmental Protection Agency's Toxic Release Inventory (TRI) project. Software tools for rigorously studying and analyzing such data could furnish both GIS maps (for understanding geospatial details and patterns about where environmental hazards are present) and (bio)chemical software for molecular visualization, or other features to help document public-health and ecological consequences, plus cleanup/remediation protocols for different classes of contaminants. Such components might be further extended by economic data profiling social impacts on surrounding communities, public health information, video and image series. etc.

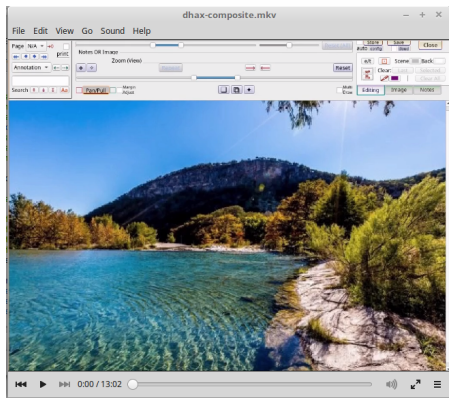
In general, then, publishers cannot assume that FAIR transparency is achieved simply by sharing raw data files. Instead, open-access materials should wherever possible include plugins or source-code extensions (developed individually for one manuscript; or collectively for a book series, journal title, or document repository) targeting science applications with requisite capabilities to view/analyze domain-specific file types. Similar comments apply to humanities environments, such as front-ends for studying linguistics corpora and/or parse-structures; sociological/economic statistics; or graphics tools for digital humanities. PacTk can help in this context because a single PacTk plugin may serve as middleware on top of which extensions could operate that are individualized to specific publications/manuscripts. This can be more efficient than building components against applications' native plugin mechanisms directly.

## ▷ Video Links

For more information, please see our software demo videos, which show software components that could be integrated into dataset applications and PDF viewers:



GIS



360° photography

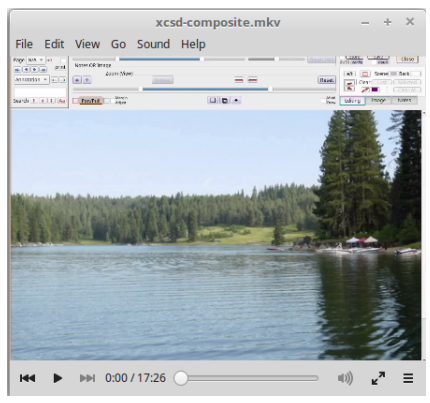


Image Processing

## *Addendum: Quotations of Interest* ▸▸▸

> Grid computing was initially conceived to enable secure access to high-end computing facilities across multiple supercomputer centers. Technologies and standards developed by the Grid computing community have evolved it into a platform to facilitate development, sharing, and integration of distributed data and analytical resources and to support applications that make use of those resources. Grid computing has been employed in a wide range of projects and applications in biomedical research. A common theme across these different application projects is the need for support for query, retrieval, and integration of data from heterogeneous resources, discovery of relevant resources, and analysis of data using one or more analytical services.
>
> "caGrid 1.0: An Enterprise Grid Infrastructure for Biomedical Research" (2008)
> https://pmc.ncbi.nlm.nih.gov/articles/PMC2274794/

> Researchers cannot readily search across multiple repositories to find models of interest, and when a researcher does retrieve a relevant model, they must determine whether it (or part of it) can be repurposed for use in their modeling work. Similarity measures and pattern-matching algorithms can help identify relevant modeling components, but existing proposals for cross-repository search and retrieval are mostly theoretical and not yet applicable in practice. Applying semantic annotations in a standardized way across model repositories would provide a common ground for such cross-repository searches, allowing models encoded in diverse formats to be retrieved based on their shared semantic descriptions.
>
> The amount of time required to select a model for reuse increases when the researcher must choose from a large set of potentially usable models. With only limited means to compare models automatically, researchers must manually assess the content, scope and underlying assumptions of each candidate model as well as the biological questions each was designed to answer. While semantic annotations cannot yet capture the purpose for which a model was built or modified, they can make the biological content of a model explicit and therefore help researchers decide whether or not to repurpose it....
>
> An additional impediment to model reuse is the lack of integration between model repositories and online collections of experimental data. If such integration existed, modelers could readily find data appropriate for validating or constraining the models they repurpose, and experimentalists could find models for use in data analysis pipelines.
>
> "Harmonizing semantic annotations for computational models in biology" (2018)
> https://pmc.ncbi.nlm.nih.gov/articles/PMC6433895/

> **Call to action**
>
> Though the full text of many scientific papers are available to researchers through CORD-19, a number of challenges prevent easy application of NLP and text mining techniques to these papers. First, the primary distribution format of scientific papers — PDF — is not amenable to text processing. The PDF file format is designed to share electronic documents rendered faithfully for reading and printing, and mixes visual with semantic information. Significant effort is needed to coerce PDF into a format more amenable to text mining, such as JATS XML, BioC (Comeau et al., 2019), or S2ORC JSON ... [W]e can still benefit from better PDF parsing tools for scientific documents. As a complement, scientific papers should also be made available in a structured format like JSON, XML, or HTML.
>
> "CORD-19: The COVID-19 Open Research Dataset" (2020)
> https://aclanthology.org/2020.nlpcovid19-acl.1.pdf

A simulation via CHASTE (Cancer, Heart And Soft Tissue Environment). An indexed grid hosting these simulation files would be able to read annotations from a variery of sources in the C++ code, including the **MultipleCryptGeometryBoundaryCondition** class (a subtype of **AbstractCellPopulationBoundaryCondition**, which is templated on the count of spatial dimensions in the model) that is specific to this model, and the **OffLatticeSimulation** class (similarly templated) within the CHASTE core, implementing the basic quantitative logic.

Likewise, the steps needed to create the video sampled and referenced in the publication should be identified. If CHASTE had included a video player component, this could be registered as a GUI asset (in practice CHASTE videos are derived externally via command-line tools, but that pipeline could be reimplemented by a host grid). A PacTk grid would program a microcomponent specifically designed to show CHASTE videos and enable context menus on paused movie frames to simulation data, parameters, provenance, and publication links.

Analogous cross-annotation schemes could be put into practice for sources potentially distributed alongside CHASTE, such as libCellML and libSBML.



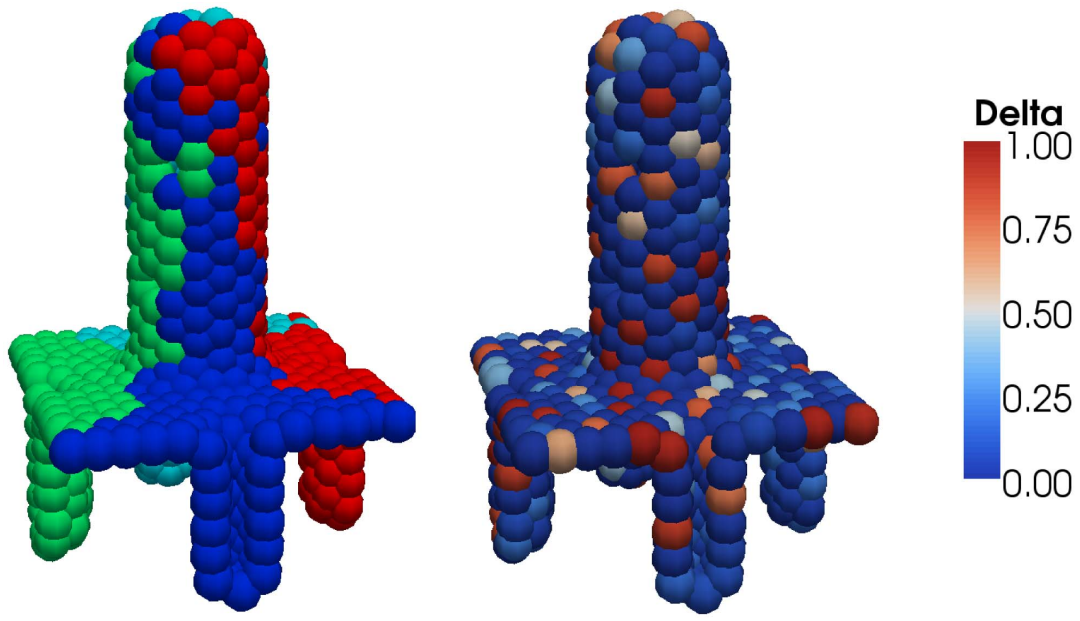



**Figure 2. 3D off-lattice simulation confined to a 2D surface: small intestinal crypts and villus.** Left: cells are labelled according to their ancestor cell; each crypt gives rise to a monoclonal population, with a multiclonal villus comprised of cells from each crypt. Right: the same simulation, here with cells labelled according to Delta levels (non-dimensionalised); Delta-Notch patterning occurs due to a signalling model inside each cell, which depends on the activity of neighbouring cells, and is thought to lead to differentiation into secretory and absorbative cell types. See also Video S2.
doi:10.1371/journal.pcbi.1002970.g002

Sources: "Chaste: An Open Source C++ Library for Computational Physiology and Biology" (https://pmc.ncbi.nlm.nih.gov/articles/PMC3597547/pdf/pcbi.1002970.pdf); and the supplemental data files at https://pmc.ncbi.nlm.nih.gov/articles/instance/3597547/bin/pcbi.1002970.s001.zip.

Figure 13: Potential locations for code/data/multimedia cross-reference annotations


On this page

Definition
Chapters and Articles
Related Terms
Recommended
Publications

FASTA

(which stands for Fast-All, a genomics format),

SRA

(sequence read archive, for

DNA

sequencing),

PDB

(Protein Data Bank, representing the

3D

geometry of protein molecules),

MAP

(electron microscopy map),

EPS

(Embedded Postscript), and

CSV

(comma-separated values). There are also tables represented in Microsoft Word or Excel formats. Although these various formats are reasonable for storing raw data, not all of them are actually machine-readable; in particular, the

EPS

, Word, and Excel files need manual processing to use the information they provide in a computational manner. A properly curated data collection would need to unify disparate sources into a common machine-readable representation (such as

XML

).


On this page

Definition
Chapters and Articles
Related Terms
Recommended
Publications

a. Enter the text shown in A1. Select A2:A501 (holding down

Shift

while tapping

PgDn

or

↓

will speed things up). Enter =RANDBETWEEN(1,100) and commit with

Ctrl

+

↵

(no

Shift

, this time). Select cell A1 and use

Ctrl

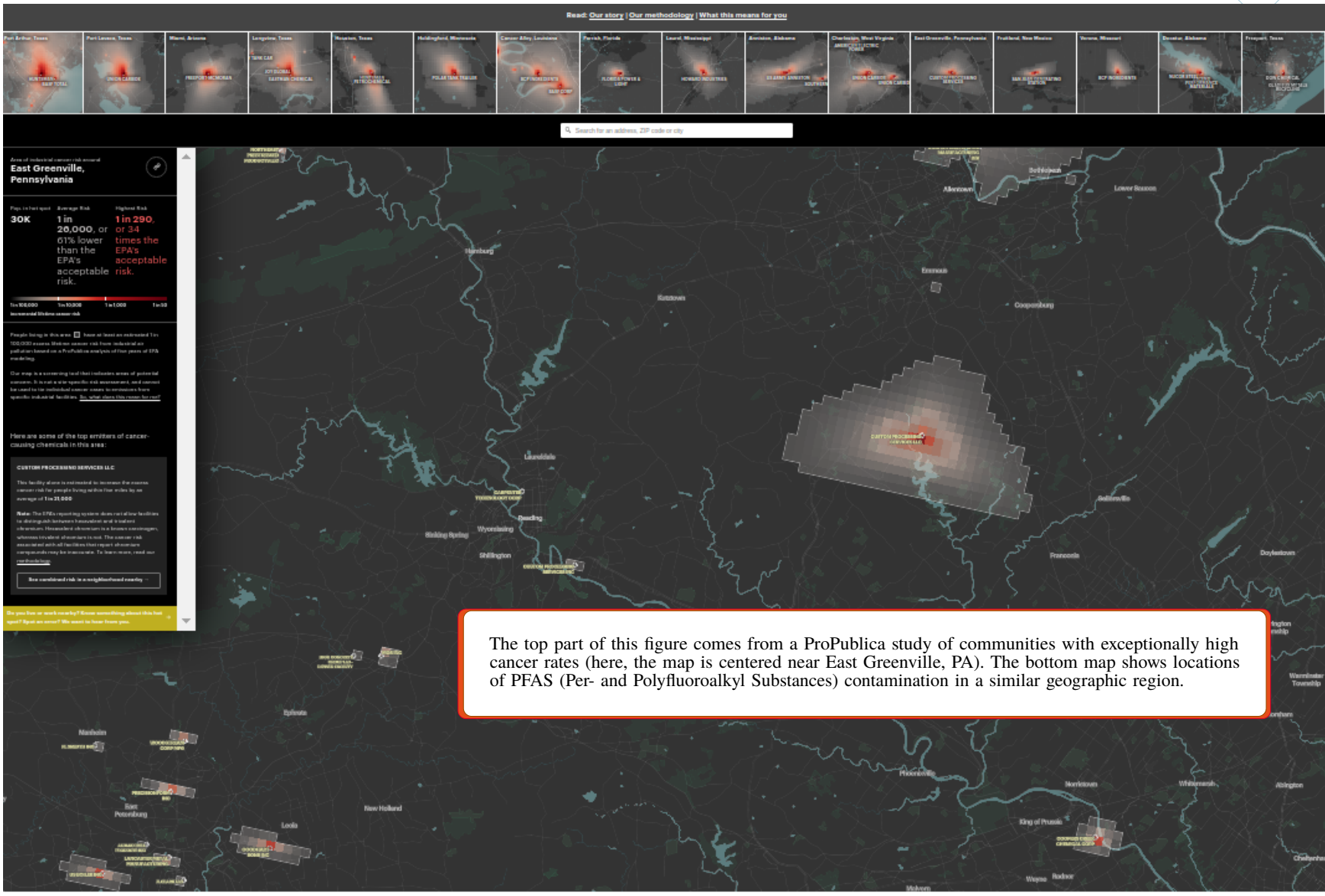
+A to highlight A1:A501 and use the *Copy* command (or

Ctrl


**Limited Support for Research Data-Mining** Even though many papers in **CORD-19** are paired with published data sets, there is currently no tool for locating research *data* through **CORD-19**. For example, the collection of manuscripts available through the Springer Nature portal linked from **CORD-19** includes over 30 **COVID-19** data sets, but researchers can only discover that these data sets exist by looking for a "supplemental materials" or a "data availability" addendum near the end of each article. These Springer Nature data sets encompass a wide array of file types and formats, including **FASTA** (which stands for Fast-All, a genomics format), **SRA** (Sequence Read Archive, for **DNA** sequencing), **PDB** (Protein Data Bank, representing the **3D** geometry of protein molecules), **MAP** (Electron Microscopy Map), **EPS** (Embedded Postscript), **CSV** (comma-separated values), and tables represented in Microsoft Word and Excel formats. To make this data more readily accessible in the context of **CORD-19**, it would be appropriate to (1) maintain an index of data sets linked to **CORD-19** articles and (2) merge these resources into a common representation (such as **XML**) wherever possible. This research-data curation can then be treated as a supplement to text-mining operations. In particular, queries against the full-text publications could be evaluated *also* as queries against the relevant set collection of research data sets.

Two book excerpts showing inaccurate HTML generation from intermediate files whose primary purpose is to produce PDF documents (① and ②). The common problem visible in the two examples is that textual or character segments which should be treated as horizontal units (subject to glue-spacing algorithms vis-à-vis words to their left and right) are, incorrectly, isolated on their own line, as if they were “vertical mode” boxes instead. Such errors naturally disrupt the surrounding lines (ones that have normal content instead). The proper way to address these issues is to specify project-specific HTML encoding rules for non-standard tag entities and/or sentence-level (or smaller) objects, rather than relying on relying on generic HTML generators. The excerpt at the bottom of the screenshot shows a correct rendering of the left-hand passage (③).

Figure 14: HTML Transcription Errors




The top part of this figure comes from a ProPublica study of communities with exceptionally high cancer rates (here, the map is centered near East Greenville, PA). The bottom map shows locations of PFAS (Per- and Polyfluoroalkyl Substances) contamination in a similar geographic region.

The specific subject-matter of environmental pollution and its relation to cancer incidents gives rise to software requirements which, ideally, are more specific than basic GIS functionality. Each geospatial area can be correlated with both cancer diagnoses (potentially with more specific tumor-classification data) and reported or measured air or ground contamination. The desire to connect these two sources of information has implications for context menus, formation of API queries, supplemental visualization windows, and other coding facets suited for domain-specific microcomponent programming.

Figure 15: GIS/Public Health Interoperability