Clerk: Moldable Live Programming for Clojure

Martin Kavalar
martin@nextjournal.com
nextjournal
Berlin, Germany

Philippa Markovics
philippa@nextjournal.com
nextjournal
Berlin, Germany

Jack Rusher
jack@nextjournal.com
nextjournal
Berlin, Germany

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Martin Kavalar, Philippa Markovics, and Jack Rusher . 2018. Clerk: Moldable Live Programming for Clojure. In *Proceedings of <Programming> 2023 (PX 23).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION: LITERATE PROGRAMMING, NOTEBOOKS AND INTERACTIVE DEVELOPMENT

Knuth's *Literate Programming* emphasized the importance of focusing on human beings as consumers of computer programs. His original implementation involved authoring files that combine source code and documentation, which were then divided into two derived artifacts: source code for the computer and a typeset document in natural language to explain the program.

At the same time, other software was developed to target scientific use cases rather than program documentation. These systems, which prefigured modern computational notebooks, ranged from REPL-driven approaches like Macsyma and Mathematica to integrated WYSIWYG editors like Ron Avitzur's *Milo*, PARC's *Tioga* and *Camino Real*, and commercial software like *MathCAD*.

In contemporary data science and software engineering practice, we often see interfaces that combine these two approaches, like Jupyter and Observable. In these notebooks, a user can mix prose, code, and visualizations in a single document that provides the advantages of Knuth's Literate Programming with those of a scientific computing environment. Unfortunately, most such systems require the programmer to use a browser-based editing environment (which alienates programmers with a strong investment in their own tooling) and custom file formats (which cause problems for integration with broader software engineering practices).

Although notebooks of this kind present an improvement on the programming experience of many languages, they often feel like a step backward to experienced Lisp programmers. In Lisp environments, it is common to be able to place the cursor after a single Lisp form and evaluate it in the context of a running program, providing finer granularity of control compared to the per-cell model of most notebooks. This workflow leads to a development style that these programmers are loath to lose.

> That LISP users tend to prefer structured growth rather than stepwise refinement is not an effect of the programming system, since both methods are supported. I believe, however, that it is a natural consequence of the interactive development method, since programs in early stages of growth can be executed and programs in early stages of refinement cannot.
> – Erik Sandewall

At the same time, though a number of Lisp environments have included graphical presentations of program objects, the default Clojure development experience relies on text-based representations of evaluation output and doesn't include the ability to embed widgets for direct manipulation of program state. Additionally, problems often arise when printing structurally large results, which can cause editor performance to degrade or lead to the truncation of output, and there's limited room for customization or support for requesting more data.

In comparison, interactive programming in Smalltalk-based systems has included GUI elements since the beginning, and work to further improve programmer experience along these lines has continued in Smalltalk-based systems like Self, Pharo, Glamorous Toolkit and Newspeak, which offer completely open and customizable integrated programming environments. Glamorous Toolkit, in particular, champions the idea of using easily constructed custom tools to improve productivity and reduce time spent on code archeology, which is also a big inspiration for what we'll present here.

2 PROGRAMMING WITH CLERK

> In such a future working relationship between human problem-solver and computer 'clerk', the capability of the computer for executing mathematical processes would be used whenever it was needed.
> – Douglas Engelbart

PX 23, March 13–17, 2023, Tokyo, Japan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2.1 Basic Interaction: Bring-Your-Own-Editor

Clerk combines Lisp-style interactive programming with the benefits of computational notebooks, literate programming, and moldable development, all without asking programmers to abandon their favorite tools or give up their existing software engineering practices. Its design stems partially from the difficult lessons we learned after years of unsuccessfully trying to get our *own team* to use an online browser-based notebook platform that we also developed.

When working with Clerk, a split-view is typically used with a code editor next to a browser showing Clerk's representation of the same notebook, as seen in *Clerk side-by-side with Emacs*.

As shown here, our *notebooks* are just source files containing regular Clojure code. Block comments are treated as markdown text with added support for LaTeX, data visualization, and so on, while top-level forms are treated as code cells that show the result of their evaluation. This format allows us to use Clerk in the context of production code that resides in revision control. Because files decorated with these comment blocks are legal code without Clerk loaded, it they can be used in many contexts where traditional notebook-specific code cannot. This has led, among other things, to Clerk being used extensively to publish documentation for libraries that are then able to ship artifacts that have no dependency on Clerk itself.

Clerk's audience is experienced Clojure developers who are familiar with interactive development. They are able to continue programming in their accustomed style, evaluating individual forms and inspecting intermediate results, but with the added ability to `show!` a namespace/file in Clerk. A visual representation of the file is then re-computed either:

- every time the file is saved, using an an optional file watcher; or alternatively,
- via an editor hot-key that can be bound to show the current document. (The authors generally prefer the hot-key over the file watcher, as it feels more direct and gives more control over when to show something in Clerk.)

Control and configuration of Clerk primarily occurs through evaluation of Clojure forms from within the programmer's environment, rather than using outside control panels and settings. This integration with the programmer's existing tooling eases adoption and allows advanced customization of the system through code.

2.2 Fast Feedback: Caching & Incremental Computation

To keep feedback loops short, Clerk uses dependency analysis to limit recomputation to forms that haven't previously been evaluated in Clerk.

In practice this means most changes to a Clerk document are reflected instantly (within 100ms) after saving a file or hitting the keybinding to update the open document.

The caching works on the level of top-level forms. A hash is computed for each top-level form. A change to the form or one of its transitive dependencies will lead to a new hash value.

When Clerk is asked to show a notebook, it will only evaluate forms that aren't cached in one of Clerk's two caches:

- an in-memory cache stores a map of the hash of a given form to its current result. This cache is limited to the current forms of the active document.
- An on-disk-cache stores the same information but to allow the user to continue work after a restart without recomputing potentially expensive operations. Because Clojure supports lazy evaluation of potentially infinite sequences, safeguards are in place to skip caching unreasonable values.

This caching behavior can be fine-tuned (or disabled) down to the level of individual forms.

The on-disk caches use a content-addressed store where each result is stored using a filename derived from the SHA-2 hash of its contents. We use the self-describing multihash format to support future changes of the hash algorithm. Additionally, a file named after the hash of a form contains a pointer to its results filename.

This combination of immutability and indirection makes distributing the cache trivial: using last-write wins for the tiny (90 bytes) pointer files. The content-addressed result cache files are never changed and can thus be synchronized without conflict.

> While I did believe, and it has been true in practice, that the vast majority of an application could be functional, I also recognized that almost all programs would need some state. Even though the host interop would provide access to (plenty of) mutable state constructs, I didn't want state management to be the province of interop; after all, a point of Clojure was to encourage people to stop doing mutable, stateful OO. In particular I wanted a state solution that was much simpler than the inherently complex locks and mutexes approaches of the hosts for concurrency-safe state. And I wanted something that took advantage of the fact that Clojure programmers would be programming primarily with efficiently persistent immutable data.

It is idiomatic in Clojure to use boxed containers to manage mutable state. While there are several of these constructs in the language, in practice atoms are the most popular by far. An atom allows reading the current value inside it with `deref/@` and updating it's value with `swap!`.

When Clerk encounters an expression in which an atom's mutable value is being read using `deref`, it will try to compute a hash based on the value *inside* the atom at runtime, and extend the expression's static hash with it.

This extension makes Clerk's caching work naturally with idiomatic use of mutable state, and frees programmers from needing to manually opt out of caching for those expressions.

2.3 Semantic Differences from regular Clojure

Clojure uses a single-pass, whole-file compilation strategy in which each evaluated form is added to the state of the running system. One positive aspect of this approach is that manually evaluating a series of forms produces the same result as loading a file containing the same forms in the same order, which is a useful property when interactively building up a program.

A practical concern with this sort of "bottom-up" programming is that the state of the system can diverge from the state of the source file, as forms that have been deleted from the source file may

still be present in the running system. This can lead to a situation where newly written code depends on values that will not exist the next time the program runs, leading to surprising errors. To help avoid this, Clerk defaults to showing an error unless it can resolve all referenced definitions in both the runtime and the source file.

It is our goal to match the semantics of Clojure as closely as possible but as a very dynamic language, there are limits to what Clerk's analysis can handle. Here are some of the things we currently do not support:

- Multiple definitions of the same var in a file
- Setting dynamic variables using `set!`
- Dynamically altering vars using `alter-var-root`
- Temporarily redefining vars using `with-redefs`

We have included a mechanism to override Clerk's error checking in cases where the user knows that one or more of these techniques are in use.

2.4 Presentation

Clerk uses a client/server architecture. The server runs in the JVM process that hosts the user's development environment. The client executes in a web browser running an embedded Clojure interpreter.

The process of conveying a value to the client is a *presentation*, a term taken from Common Lisp systems that support similar features . The process of presentation makes use of *viewers*, each of which is a map from well-known keys to quoted forms containing source code for Clojure functions that specify how the client should render data structures of a given type. When a viewer form is received on the client side, it is compiled into a function that will be then called on data later sent by the server.

When the `present` function is called on the server side, it defaults to performing a depth-first traversal of the data structure it receives, attaching appropriate viewers at each node of the tree. The resulting structure containing both data and viewers is then sent to the client.

To avoid overloading the browser or producing uselessly large output, Clerk's built-in collection viewer carries an attribute to control the number of items initially displayed, allowing more data to be requested by the user on demand. Besides this simple limit, there's a second global *budget* per result to limit the total number of items shown in deeply nested data structures. We've found this simple system to work fairly well in practice.

One benefit of using the browser for Clerk's rendering layer is that it can produce static HTML pages for publication to the web. We could not resist the temptation to produce this document with Clerk, and have used that experience as an opportunity to improve the display of sidenotes.

It's also possible to use Clerk's presentation system in other contexts. We know of at least one case of a user leveraging Clerk's presentation system to do in-process rendering without a browser.

2.5 Built-in Viewers

Clerk comes with a set of built-in viewers for common situations. These include support for Clojure's immutable data structures, HTML (including the hiccup variant that is often used in Clojure to represent HTML and SVG), data visualization, tables, LaTeX, source code, images, and grids, as well as a fallback viewer based on Clojure's printer. The Book of Clerk gives a good overview of the available built-ins. Because Clerk's client is running in the browser, we are able to benefit from the vast JS library ecosystem. For example we're using Plotly and vega for plotting, CodeMirror for rendering code cells, and KaTeX for typesetting math.

Clerk's built-in viewers try to suit themselves to typical Data Science use cases. By default, Clerk shows a code block's result as-is with some added affordances like syntax coloring and expandability of large sub-structures that are collapsed by default.

Here is an interactive example of the well-known `iris` data set that has been added as a dependency to this notebook. Clicking the disclosure triangles will expand the data structure.

```
datasets/iris
```

Additional affordances are modes to auto-expand nested structures based on shape heuristics and expanding multiple sub-structures of the same level:

Using the built-in `clerk/table` viewer, the same data structure can also be rendered as table. The table viewer is using heuristics to infer the makeup of the table, such as column headers, from the structure of the data:

```
(clerk/table datasets/iris)
```

Together with tables, plots make up for the most common Data Science use cases. Clerk comes with built-in support for the popular vega and Plotly plotting grammars.

In the following figure, the same `iris` dataset, as shown in the above table example, is used to render an interactive `vega-lite` plot using the `clerk/vl` viewer:

```
(clerk/vl {:data {:values datasets/iris}
           :width 500
           :height 500
           :title "sepal-length vs. sepal-width"
           :mark {:type "point"
                  :tooltip {:field :species}}
           :encoding {:color {:field :species}
                      :x {:field :sepal-length
                          :type :quantitative
                          :scale {:zero false}}
                      :y {:field :sepal-width
                          :type :quantitative
                          :scale {:zero false}}}
           :embed/opts {:actions false}})
```

It is important to note that Clerk's viewers work in a way that encourages composition. Multiple viewers can be combined to suit a specific use case such as the following example showing a table of airline passenger numbers by year and quarter and embedding a sparkline graph into the table row for each year.

A typical Clerk workflow for this would be to first take a look at the shape of the data:

```
datasets/air-passengers
```

Then, a `sparkline` function is defined that generates the graph (using `clerk/vl`) to be embedded into each table row later:

```
(defn sparkline [values]
  (clerk/vl {:data {:values (map-indexed (fn [i n] {:x i :y n}) va
             :mark {:type :line :strokeWidth 1.2}
```

```
            :width 140
            :height 20
            :config {:background nil :border nil :view {:stroke :transparent}}
            :encoding {:x {:field :x :type :ordinal :axis {:title nil :labels false}}
                       :y {:field :y :type :quantitative :axis nil :scale {:type :log}}}
            :embed/opts {:actions false}}))
```

And finally reducing the data to quarters and years and adding the sparkline graphs in a final mapping step:

```
(clerk/table
 {:head ["Year" "Q1" "Q2" "Q3" "Q4" "Trend"]
  :rows (->> datasets/air-passengers
             (group-by :year)
             (map (fn [[year months]]
                    (let [qs (->> months (map :n) (partition 3) (map #(reduce + %)))]
                      (concat [year] qs [(sparkline qs)]))))
             (sort-by first))})
```

2.6 Moldable Viewer API

Clerk's viewers are an ordered (and thus prioritized) collection of plain Clojure hash maps. Clerk interprets the following optional keys in each viewer map:

- `:pred` is a predicate function that tests whether this viewer should be used for a given data structure
- `:transform-fn` is an optional function run on the server side to transform data before sending it to the client. It receives a map argument with the original value under a key. Additional keys carry the path, the viewer stack, and the budget (for elision)
- `:render-fn` is a quoted form that will be sent to the browser, where it will be compiled into a function that will be called to display the data
- `:page-size` is a number that indicates how many items to send in each chunk during elision/pagination

Viewers can also be explicitly selected by wraping a value in the `clerk/with-viewer` function, which produces a presentation for that value using that viewer. Alternatively, viewers can be selected by placing a Clojure metadata declaration before a form. Because of the way Clojure handles compilation, metadata in this position is ultimately ignored in the generated code. So far as we know, this is a novel mechanism for out of band signaling to a specialized Clojure parser. TODO add example viewer source and metadata declaration here

The process of selecting viewers happens programmatically on the server side, thus using the programmer's already existing interactive programming environment as a user interface.

2.7 Sync

To help with creating interactive tools using Clerk, it also supports bidirectional sync of state between the client and server Clojure environments. If a Clojure `atom` on the server is annotated with metadata indicating it is `sync`, Clerk will create a corresponding var in the client environment. Both of these atoms will be automatically instrumented with an update watcher that broadcasts a *diff* to the other side.

In addition, a server-side change will trigger a refresh of the currently active document, which will then re-calculate the minimum subset of the document that is dependent on that atom's value. This allows background Clerk for small local-first apps, as shown in the Basic Slitting Example.

2.8 Tap Stream Inspector

Clerk also comes with an inspector for Clojure's tap system.

> tap is a shared, globally accessible system for distributing a series of informational or diagnostic values to a set of (presumably effectful) handler functions. It can be used as a better debug prn, or for facilities like logging etc.
>
> – Clojure 1.10 Changelog

When enabled, Clerk will attach a tap listener function and record and show the tap stream. This makes Clerk's viewer system accessible across file and namespace boundaries and independently of the caching mechanisms.

2.9 Prose-oriented Documents

The first and primary use case for Clerk was adding prose, visualizations, and interactivity to Clojure namespaces. However, when writing documents that are mainly prose, but would benefit from *some* computational elements, it is rather tedious to write everything in comment blocks. To make this easier, Clerk can also operate on markdown files with "code-fenced" source code blocks. All Clojure source blocks in such a file are evaluated and replaced in the generated document with their result.

This format is very similar to other markdown-based notebooks, like R Markdown, but specifically tailored to Clojure. We used this approach to write this paper, the source for which is located on Github.

3 EXAMPLES OF MOLDABLE DEVELOPMENT WITH CLERK

In addition to the sorts of traditional data science use cases that one might expect from something that has "notebook" features, we intend Clerk to be a general purpose programmer's assistant that allows the rapid construction of tiny interfaces during daily work. Here are a few samples of tools and documentation created in this manner.

3.1 Augmenting table names

This example illustrates an approach we used to make working with a legacy DB2 database easier. The database's column names are made up of largely human-unreadable 8 character sequences:

We were able to automatically translate these names using a metaschema extracted from the database. This allowed us to create a viewer that maps those 8-character names to human-readable (German-only) names (which we can then translate to English names). In typical Lisp fashion, we go on and inspect a query interactively. We can use the translated names in the table even print them but one quickly sees the limit of plain-text printing:

With Clerk, were able to render the output as a graphical table without the limitations of plain text. Further, we can use the Viewer

API to extend the table viewer's headings to show the translated metaschema names (plus showing the original 8 character names in a de-emphasized way so that they aren't lost). We can go further still, showing the original German names when move the mouse over the headings:

3.2 Rich documentation features

This example illustrates the use of Clerk to create rich documentation for `clojure2d`'s colors package. They used Clerk's Viewer API to implement custom viewers to visualize colors, gradients and color spaces, then publish that documentation on the web by generating a static website directly from the source code of the library.

3.3 Regex Dictionary

Built as a showcase for Clerk's sync feature, this example allows entering a regex into a text input and get dictionary matches as result while you type:

It is built using a Clojure atom containing the text input's current value that is synced between the client and server. As you type into the input, the atom's content will be updated and synced. Consequently, printing the atom's content in your editor will show the input's current value:

3.4 Lurk: Interactive Lucene-powered Log Search

Also building on Clerk's sync feature, this interactive log search uses Lucene on the JVM side to index and search a large number of log entries. In addition to using query input, logs can also be filtered by timeframe via an interactive chart. It is worth noting that this example has a completely custom user interface styling (nothing left of Clerk's default styling) via Clerk's CSS customization options.

3.5 Experience

Our experience as the developers and users of Clerk has been surprisingly positive, but we're heavily biased. We've chosen a few quotes from Clerk's user base to give a sense of how it has been received in the community:

> [Clerk] is making the training of junior Clojure programmers a massive pleasure! [...]
> It helps us to bypass what would otherwise be a lot of distracting UI programming. Set up your env, make a namespace, hit a keybind, hey presto, your code is running in a browser.
> – Robert Stuttaford

> I'm using Clerk to visualize statistics properties from a simulation in a model checker [...] it's basically a wrapper over TLA+ [...]
> Amazing that Clerk just lets you focus on what really matters and nothing else!
> – Paulo Feodrippe

> I just wanted to express some gratitude for Clerk. It's been a game changer for me in terms of understanding problems and communicating that understanding to other people.
> – Jeffrey Simon

4 RELATED & FUTURE WORK

Besides the aforementioned work there's a number of contemporary related systems:

- Org mode is a major mode for Emacs supporting polyglot literate programming based on a plain text format.
- Streamlit is a Python library that eshews a custom format and enables building a web UI on regular python scripts. Its caching system memoizes functions that are tagged using Python's decorators.
- Pluto is a Julia library that uses static analysis to enable incremental computation and two-way bindings. It does come with a web-based editor. Its format are plain Julia files with comment annotations for cell ids and execution order.
- Livebook is an Elixir notebook with code editing in the browser and explicit per-cell execution. It serializes notebooks to a Markdown format.

Our goal with the development of Clerk is to *leave the toolbox open*: we want Clerk's users to be able to customize behavior, often by providing functions.

Clerk's viewer api is a first example of that but we want to take this further by letting users:

- provide functions to control the caching e.g. to support more efficient caching of data frames
- letting the viewer api's `:pred` function opt into receiving more context like the path in the tree
- make caching more granular and support caching function invocations
- override `parse` and `eval` to support different syntaxes than markdown and different semantics

So far we've mainly used Clerk's caching on local machines in isolation. We plan to share a distributed cache within our dev team in order to learn about the benefits and challenges this can bring. We also want to extend Clerk to better communicate caching behavior to its users (why a value could or could not be cached, if it was cached in memory or on-disk).

We've been talking about ways to write changes originating from controls in Clerk's view back to the source files. We also believe that for this to be a good developer experience, concurrent modifications without intermediate saving should be supported. Making a simple integration that works on level of source files insufficient. Since this is a significant chunk of work and will require a different solution for each editor, we've avoided it until now.

5 CONCLUSION

□

---

6 INTRODUCTION

ACM's consolidated article template, introduced in 2017, provides a consistent LaTeX style for use across ACM publications, and incorporates accessibility and metadata-extraction functionality necessary for future Digital Library endeavors. Numerous ACM and

SIG-specific LaTeX templates have been examined, and their unique features incorporated into this single new template.

If you are new to publishing with ACM, this document is a valuable guide to the process of preparing your work for publication. If you have published with ACM before, this document provides insight and instruction into more recent changes to the article template.

The "acmart" document class can be used to prepare articles for any ACM publication — conference or journal, and for any stage of publication, from review to final "camera-ready" copy, to the author's own version, with *very* few changes to the source.

7 TEMPLATE OVERVIEW

As noted in the introduction, the "acmart" document class can be used to prepare many different kinds of documentation — a double-blind initial submission of a full-length technical paper, a two-page SIGGRAPH Emerging Technologies abstract, a "camera-ready" journal article, a SIGCHI Extended Abstract, and more — all by selecting the appropriate *template style* and *template parameters*.

This document will explain the major features of the document class. For further information, the *LaTeX User's Guide* is available from https://www.acm.org/publications/proceedings-template.

7.1 Template Styles

The primary parameter given to the "acmart" document class is the *template style* which corresponds to the kind of publication or SIG publishing the work. This parameter is enclosed in square brackets and is a part of the documentclass command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the acmsmall template style:

- acmsmall: The default journal template style.
- acmlarge: Used by JOCCH and TAP.
- acmtog: Used by TOG.

The majority of conference proceedings documentation will use the acmconf template style.

- acmconf: The default proceedings template style.
- sigchi: Used for SIGCHI conference articles.
- sigplan: Used for SIGPLAN conference articles.

7.2 Template Parameters

In addition to specifying the *template style* to be used in formatting your work, there are a number of *template parameters* which modify some part of the applied template style. A complete list of these parameters can be found in the *LaTeX User's Guide*.

Frequently-used parameters, or combinations of parameters, include:

- anonymous,review: Suitable for a "double-blind" conference submission. Anonymizes the work and includes line numbers. Use with the command to print the submission's unique ID on each page of the work.
- authorversion: Produces a version of the work suitable for posting by the author.
- screen: Produces colored hyperlinks.

This document uses the following string as the first command in the source file:

```
\documentclass[sigconf]{acmart}
```

8 MODIFICATIONS

Modifying the template — including but not limited to: adjusting margins, typeface sizes, line spacing, paragraph and list definitions, and the use of the \vspace command to manually adjust the vertical spacing between elements of your work — is not allowed.

Your document will be returned to you for revision if modifications are discovered.

9 TYPEFACES

The "acmart" document class requires the use of the "Libertine" typeface family. Your TeX installation should include this set of packages. Please do not substitute other typefaces. The "lmodern" and "ltimes" packages should not be used, as they will override the built-in typeface families.

10 TITLE INFORMATION

The title of your work should use capital letters appropriately - https://capitalizemytitle.com/ has useful rules for capitalization. Use the title command to define the title of your work. If your work has a subtitle, define it with the subtitle command. Do not insert line breaks in your title.

If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text. The title command has a "short title" parameter:

```
\title[short title]{full title}
```

11 AUTHORS AND AFFILIATIONS

Each author must be defined separately for accurate metadata identification. As an exception, multiple authors may share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors' e-mail addresses whenever possible.

Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

```
\author{Brooke Aster, David Mehldau}
\email{dave,judy,steve@university.edu}
\email{firstname.lastname@phillips.org}
```

The authornote and authornotemark commands allow a note to apply to multiple authors — for example, if the first two authors of an article contributed equally to the work.

If your author list is lengthy, you must define a shortened version of the list of authors to be used in the page headers, to prevent overlapping text. The following command should be placed just after the last \author{} definition:

```
\renewcommand{\shortauthors}{McCartney, et al.}
```

Omitting this command will force the use of a concatenated list of all of the authors' names, which may result in overlapping text in the page headers.

The article template's documentation, available at https://www.acm.org/publications/proceedings-template, has a complete explanation of these commands and tips for their effective use.

Note that authors' addresses are mandatory for journal articles.

12 RIGHTS INFORMATION

Authors of any work published by ACM will need to complete a rights form. Depending on the kind of work, and the rights management choice made by the author, this may be copyright transfer, permission, license, or an OA (open access) agreement.

Regardless of the rights management choice, the author will receive a copy of the completed rights form once it has been submitted. This form contains LaTeX commands that must be copied into the source document. When the document source is compiled, these commands and their parameters add formatted text to several areas of the final document:

- the "ACM Reference Format" text on the first page.
- the "rights management" text on the first page.
- the conference information in the page header(s).

Rights information is unique to the work; if you are preparing several works for an event, make sure to use the correct set of commands with each of the works.

The ACM Reference Format text is required for all articles over one page in length, and is optional for one-page articles (abstracts).

13 CCS CONCEPTS AND USER-DEFINED KEYWORDS

Two elements of the "acmart" document class provide powerful taxonomic tools for you to help readers find your work in an online search.

The ACM Computing Classification System — https://www.acm.org/publications/class-2012 — is a set of classifiers and concepts that describe the computing discipline. Authors can select entries from this classification system, via https://dl.acm.org/ccs/ccs.cfm, and generate the commands to be included in the LaTeX source.

User-defined keywords are a comma-separated list of words and phrases of the authors' choosing, providing a more flexible way of describing the research being presented.

CCS concepts and user-defined keywords are required for for all articles over two pages in length, and are optional for one- and two-page articles (or abstracts).

14 SECTIONING COMMANDS

Your work should use standard LaTeX sectioning commands: `section`, `subsection`, `subsubsection`, and `paragraph`. They should be numbered; do not remove the numbering from the commands.

Simulating a sectioning command by setting the first word or words of a paragraph in boldface or italicized text is **not allowed.**

15 TABLES

The "`acmart`" document class includes the "`booktabs`" package — https://ctan.org/pkg/booktabs — for preparing high-quality tables.

Table captions are placed *above* the table.

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material are found in the *LaTeX User's Guide.*

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
ϕ	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed output of this document.

Always use midrule to separate table header rows from data rows, and use it only for this purpose. This enables assistive technologies to recognise table headers and support their users in navigating tables more easily.

16 MATH EQUATIONS

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

16.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin ... \end` construction or with the short form `$...$`. You can use any of the symbols and structures, from α to ω, available in LaTeX [24]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

16.2 Display Equations

A numbered display equation—one set off by vertical space from the text and centered horizontally—is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Table 2: Some Typical Commands

Command	A Number	Comments
`\author`	100	Author
`\table`	300	For tables
`\table*`	400	For wider tables

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

17 FIGURES

The "`figure`" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

Your figures should contain a caption which describes the figure to the reader.

Figure captions are placed *below* the figure.

Every figure should also have a figure description unless it is purely decorative. These descriptions convey what's in the image to someone who cannot see it. They are also used by search engine crawlers for indexing images, and when images cannot be loaded.

A figure description must be unformatted plain text less than 2000 characters long (including spaces). **Figure descriptions should not repeat the figure caption – their purpose is to capture important information that is not already provided in the caption or the main text of the paper.** For figures that convey important and complex new information, a short text description may not be adequate. More complex alternative descriptions can be placed in an appendix and referenced in a short figure description. For example, provide a data table capturing the information in a bar chart, or a structured list representing a graph. For additional information regarding how best to write figure descriptions and why doing this is so important, please see https://www.acm.org/publications/taps/describing-figures/.

17.1 The "Teaser Figure"

A "teaser figure" is an image, or set of images in one figure, that are placed after all author and affiliation information, and before the body of the article, spanning the page. If you wish to have such a figure in your article, place the command immediately before the `\maketitle` command:

```
\begin{teaserfigure}
  \includegraphics[width=\textwidth]{sampleteaser}
  \caption{figure caption}
  \Description{figure description}
\end{teaserfigure}
```

18 CITATIONS AND BIBLIOGRAPHIES

The use of BibTeX for the preparation and formatting of one's references is strongly recommended. Authors' names should be complete — use full first names ("Donald E. Knuth") not initials ("D. E. Knuth") — and the salient identifying features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

The bibliography is included in your source document with these two commands, placed just before the `\end{document}` command:

```
\bibliographystyle{ACM-Reference-Format}
\bibliography{bibfile}
```

where "`bibfile`" is the name, without the "`.bib`" suffix, of the BibTeX file.

Citations and references are numbered by default. A small number of ACM publications have citations and references formatted in the "author year" style; for these exceptions, please include this command in the **preamble** (before the command "`\begin{document}`") of your LaTeX source:

```
\citestyle{acmauthoryear}
```

Some examples. A paginated journal article [2], an enumerated journal article [10], a reference to an entire issue [9], a monograph (whole book) [23], a monograph/whole book in a series (see 2a in spec. document) [17], a divisible-book such as an anthology or compilation [12] followed by the same example, however we only output the series if the volume number is given [13] (so Editor00a's series should NOT be present since it has no vol. no.), a chapter in a divisible book [35], a chapter in a divisible book in a series [11], a multi-volume work as book [22], a couple of articles in a proceedings (of a conference, symposium, workshop for example) (paginated proceedings article) [3, 15], a proceedings article with all possible elements [34], an example of an enumerated proceedings article [14], an informally published work [16], a couple of preprints [6, 7], a doctoral dissertation [8], a master's thesis: [4], an online document / world wide web resource [1, 28, 36], a video game (Case 1) [27] and (Case 2) [26] and [25] and (Case 3) a patent [33], work accepted for publication [30], 'YYYYb'-test for prolific author [31] and [32]. Other cites might contain 'duplicate' DOI and URLs (some SIAM articles) [21]. Boris / Barbara Beeton: multi-volume works as books [19] and [18]. A couple of citations with DOIs: [20, 21]. Online citations: [36–38]. Artifacts: [29] and [5].

19 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

This section has a special environment:

```
\begin{acks}
...
\end{acks}
```

so that the information contained therein can be more easily collected during the article metadata extraction phase, and to ensure consistency in the spelling of the section heading.

Authors should not prepare this section as a numbered or unnumbered `\section`; please use the "acks" environment.

20 APPENDICES

If your work needs an appendix, add it before the "`\end{document}`" command at the conclusion of your source document.

Start the appendix with the "`appendix`" command:

```
\appendix
```

and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating the section and subsection identification method.

21 MULTI-LANGUAGE PAPERS

Papers may be written in languages other than English or include titles, subtitles, keywords and abstracts in different languages (as a rule, a paper in a language other than English should include an English title and an English abstract). Use `language=...` for every language used in the paper. The last language indicated is the main language of the paper. For example, a French paper with additional titles and abstracts in English and German may start with the following command

```
\documentclass[sigconf, language=english, language=german,
               language=french]{acmart}
```

The title, subtitle, keywords and abstract will be typeset in the main language of the paper. The commands `\translatedXXX`, `XXX` begin title, subtitle and keywords, can be used to set these elements in the other languages. The environment `translatedabstract` is used to set the translation of the abstract. These commands and environment have a mandatory first argument: the language of the second argument. See `sample-sigconf-i13n.tex` file for examples of their usage.

22 SIGCHI EXTENDED ABSTRACTS

The "`sigchi-a`" template style (available only in LaTeX and not in Word) produces a landscape-orientation formatted article, with a wide left margin. Three environments are available for use with the "`sigchi-a`" template style, and produce formatted output in the margin:

`sidebar`: Place formatted text in the margin.
`marginfigure`: Place a figure in the margin.
`margintable`: Place a table in the margin.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] Rafal Ablamowicz and Bertfried Fauser. 2007. *CLIFFORD: a Maple 11 Package for Clifford Algebra Computations, version 11.* Retrieved February 28, 2008 from http://math.tntech.edu/rafal/cliff11/index.html

[2] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. https://doi.org/10.1145/1188913.1188915

[3] Sten Andler. 1979. Predicate Path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL '79).* ACM Press, New York, NY, 226–236. https://doi.org/10.1145/567752.567774

[4] David A. Anisi. 2003. *Optimal Motion Control of a Ground Vehicle.* Master's thesis. Royal Institute of Technology (KTH), Stockholm, Sweden.

[5] Sam Anzaroot and Andrew McCallum. 2013. *UMass Citation Field Extraction Dataset.* Retrieved May 27, 2019 from http://www.iesl.cs.umass.edu/data/data-umasscitationfield

[6] Sam Anzaroot, Alexandre Passos, David Belanger, and Andrew McCallum. 2014. Learning Soft Linear Constraints with Application to Citation Field Extraction. arXiv:1403.1349

[7] Lutz Bornmann, K. Brad Wray, and Robin Haunschild. 2019. Citation concept analysis (CCA)—A new form of citation analysis revealing the usefulness of concepts for other researchers illustrated by two exemplary case studies including classic books by Thomas S. Kuhn and Karl R. Popper. arXiv:1905.12410 [cs.DL]

[8] Kenneth L. Clarkson. 1985. *Algorithms for Closest-Point Problems (Computational Geometry).* Ph. D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171.

[9] Jacques Cohen (Ed.). 1996. Special issue: Digital Libraries. *Commun. ACM* 39, 11 (Nov. 1996).

[10] Sarah Cohen, Werner Nutt, and Yehoshua Sagic. 2007. Deciding equivalances among conjunctive aggregate queries. *J. ACM* 54, 2, Article 5 (April 2007), 50 pages. https://doi.org/10.1145/1219092.1219093

[11] Bruce P. Douglass, David Harel, and Mark B. Trakhtenbrot. 1998. Statecarts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. https://doi.org/10.1007/3-540-65193-4_29

[12] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. https://doi.org/10.1007/3-540-09237-4

[13] Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100. https://doi.org/10.1007/3-540-09237-4

[14] Matthew Van Gundy, Davide Balzarotti, and Giovanni Vigna. 2007. Catch me, if you can: Evading network signatures with web-based polymorphic worms. In *Proceedings of the first USENIX workshop on Offensive Technologies (WOOT '07).* USENIX Association, Berkley, CA, Article 7, 9 pages.

[15] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. 1993. Maintaining Discrete Probability Distributions Optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 700).* Springer-Verlag, Berlin, 253–264.

[16] David Harel. 1978. *LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER.* MIT Research Lab Technical Report TR-200. Massachusetts Institute of Technology, Cambridge, MA.

[17] David Harel. 1979. *First-Order Dynamic Logic.* Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. https://doi.org/10.1007/3-540-09237-4

[18] Lars Hörmander. 1985. *The analysis of linear partial differential operators. III.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. viii+525 pages. Pseudodifferential operators.

[19] Lars Hörmander. 1985. *The analysis of linear partial differential operators. IV.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. vii+352 pages. Fourier integral operators.

[20] IEEE 2004. IEEE TCSC Executive Committee. In *Proceedings of the IEEE International Conference on Web Services (ICWS '04).* IEEE Computer Society, Washington, DC, USA, 21–22. https://doi.org/10.1109/ICWS.2004.64

[21] Markus Kirschmer and John Voight. 2010. Algorithmic Enumeration of Ideal Classes for Quaternion Orders. *SIAM J. Comput.* 39, 5 (Jan. 2010), 1714–1747. https://doi.org/10.1137/080734467

[22] Donald E. Knuth. 1997. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd. ed.).* Addison Wesley Longman Publishing Co., Inc.

[23] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY.

[24] Leslie Lamport. 1986. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, MA.
[25] Newton Lee. 2005. Interview with Bill Kinder: January 13, 2005. Video. *Comput. Entertain.* 3, 1, Article 4 (Jan.-March 2005). https://doi.org/10.1145/1057270.1057278
[26] Dave Novak. 2003. Solder man. Video. In *ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27–27, 2003).* ACM Press, New York, NY, 4. https://doi.org/99.9999/woot07-S422 http://video.google.com/videoplay?docid=6528042696351994555
[27] Barack Obama. 2008. A more perfect union. Video. Retrieved March 21, 2008 from http://video.google.com/videoplay?docid=6528042696351994555
[28] Poker-Edge.Com. 2006. Stats and Analysis. Retrieved June 7, 2006 from http://www.poker-edge.com/stats.php
[29] R Core Team. 2019. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
[30] Bernard Rous. 2008. The Enabling of Digital Libraries. *Digital Libraries* 12, 3, Article 5 (July 2008). To appear.
[31] Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. 2010. A library-based synthesis methodology for reversible logic. *Microelectron. J.* 41, 4 (April 2010), 185–194.
[32] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. 2010. Synthesis of Reversible Circuit Using Cycle-Based Approach. *J. Emerg. Technol. Comput. Syst.* 6, 4 (Dec. 2010).
[33] Joseph Scientist. 2009. The fountain of youth. Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.
[34] Stan W. Smith. 2010. An experiment in bibliographic mark-up: Parsing metadata for XML export. In *Proceedings of the 3rd. annual workshop on Librarians and Computers (LAC '10, Vol. 3)*, Reginald N. Smythe and Alexander Noble (Eds.). Paparazzi Press, Milan Italy, 422–431. https://doi.org/99.9999/woot07-S422
[35] Asad Z. Spector. 1990. Achieving application requirements. In *Distributed Systems* (2nd. ed.), Sape Mullender (Ed.). ACM Press, New York, NY, 19–33. https://doi.org/10.1145/90417.90738
[36] Harry Thornburg. 2001. *Introduction to Bayesian Statistics.* Retrieved March 2, 2005 from http://ccrma.stanford.edu/~jos/bayes/bayes.html
[37] TUG 2017. *Institutional members of the TeX Users Group.* Retrieved May 27, 2017 from http://wwtug.org/instmem.html
[38] Boris Veytsman. 2017. *acmart—Class for typesetting publications of ACM.* Retrieved May 27, 2017 from http://www.ctan.org/pkg/acmart

A RESEARCH METHODS

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009