International
Virtual
Observatory
Alliance

The IvoaTEX Document Preparation System

Version 1.3

IVOA Note 2022-06-14

Working Group
Standards and Processes
This version
https://www.ivoa.net/documents/ivoatexDoc/20220614
Latest version
https://www.ivoa.net/documents/ivoatexDoc
Previous versions
Version 1.1
Version 1.0
Author(s)
Markus Demleitner, Mark Taylor, Paul Harrison, Marco Molinaro
Editor(s)
Markus Demleitner
Version Control
Revision 769b2a3, 2022-05-11 11:53:51 +0200

Abstract

This note describes the IvoaTEX document preparation system for IVOA standards and notes. IvoaTEX supports the production of PDF and HTML renderings of the documents with sources in plain text suitable for version control, as is desirable for normative texts. This note contains a user guide as well as a discussion of IvoaTEX's dependencies and its implementation. It describes the software as of May 2022 (release 1.2). While IvoaTEX is not usually explicitly installed, it can be downloaded from `http://ivoatex.g-vo.org`.

Status of this document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found at https://www.ivoa.net/documents/.

Contents

Acknowledgments

IVOATEX heavily draws from experiences made with previous markup-based document preparation systems, in particular LaTeX classes and infrastructure created by Sébastien Derriere and Mark Taylor, as well as Paul Harrison's XML-based ivoadoc system.

We thank tth's author, Ian Hutchinson, for generous technical support and prompt provision of solutions in the upstream source where necessary.

1 Introduction

Creating and developing standards is a big part of the operations of the International Virtual Observatory Alliance (IVOA). As these are normative texts, attention to detail is very important, and being able to rigorously track changes to the documents is highly advantageous.

Standards are also often developed cooperatively, which means that capabilities for branching and merging are desirable. This strongly suggests employing version control systems for document authoring. Change tracking

in software designed for editing office documents, to the extent it is supported at all, usually requires significant manual intervention, is optional, often used incorrectly, and frequently lacks interoperability. Led by these considerations, it was decided that IVOATEX would have to be based on plain text source files.

As mandated by the IVOA Document Standards (Genova and Arviset et al., 2017), finished documents have to be at least available in PDF, while an additional HTML rendering for online use is recommended. A document preparation system should thus be able to produce documents in these formats in at least acceptable quality.

With these constraints in mind, several possible solutions were investigated. Paul Harrison's ivoadoc system[1] went for XHTML as an input format and used XSLT2 and XML-FO as document processors. While this facilitated several interesting features – for instance, automatic extraction and formatting of XML schema fragments or straightforward embedding of RDFa markup for machine-readable examples –, it turned out that tooling issues were severe (e.g., reliable use of SGML catalogs[2], non-free hyphenation patterns, classpath issues) and the use of XML-FO for PDF generation yielded inferior renderings with little prospect for improvements by third parties. Also, authors disliked writing HTML tags.

Other options considered for source languages included docbook or a lightweight markup language (ReStructuredText and markdown having been serious contenders). In each case, there were concerns either regarding the system's power and flexibility or its ease of installation and maintenance.

Meanwhile, several documents – to mention just a few, SAMP, VOTable, and VOUnits – had successfully used TEX-based systems typically derived from work done in the early 2000s by Sébastien Derriere. IVOATEX essentially is a generalisation of these standards' formatting systems, also inheriting from them the use of the `make` tool to automate workflows.

IVOATEX was extended to relieve document editors from some of the bookkeeping involved with producing IVOA standards and provide authors with uniform solutions for common problems in standards typesetting.

In the remainder of this document, we give quick-start instructions on installation and authoring in sect. 2 and continue with a more thorough discussion of IVOATEX's facilities, focusing on enabling automatic production of both HTML and PDF output in sect. 3. Added in this document's version 1.2, sect. 4 then documents how ivoatex is intended to be used on github by the various actors; authors and contributors to IVOA standards should at least have a brief look at the recommended procedures for them in subsec-

[1] `https://volute.g-vo.org/svn/trunk/projects/ivoapub/ivoadoc`

[2] This is important as retrieval of DTDs and similar data from their commonly used system identifiers (i.e., typically W3C web servers) is at least undesirable and in practice causes massive delays in formatting due to rate limiting on the part of the W3C.

tion 4.2. In sect. 5, additional details on the implemenation are given for the benefit of authors planning to extend IvoaTeX. We close with a discussion of open issues and desirable developments.

2 Installation and Quick Start

2.1 Dependencies

IvoaTeX is designed to work more or less out of the box on common POSIX-compliant systems; no non-free software is required for operation. Its main uncommon dependency is the tth translator, a program based on lex used to translate LaTeX to HTML. As it is compact and portable, it is delivered with IvoaTeX and built on demand. Since IvoaTeX's tth may at times offer some enhancements over the upstream tth, using a system-installed tth is discouraged.

The remaining dependencies include:

- A LaTeX distribution with some commonly available packages (calc, graphicx, xcolor, ifthen, doc, paralist, url, natbib, caption, hyperref). It is recommended to install TeXLive.
- A sufficiently capable implementation of `make`, with GNU's implementation recommended.
- `librsvg2-bin` when people need to (re-) build architecture diagrams.
- Optionally, `latexmk`. If it is available, a lot less manual attention is necessary when rerunning LaTeX to update bibliographies or document-internal references.
- `python3` for generated content, and other housekeeping that is probably not relevant for most authors. Editors need it for automatic submission to the document repository.
- Editors will need the XSLT1 processor `xsltproc` (a different processor can be used, but that would probably require custom make rules) for HTML generation.
- Editors will need the `gcc` compiler (another C compiler could be used; the central makefile should probably be amended to allow easier changes here), and `flex` for HTML generation.
- Editors will need the `zip` archiver for package generation.
- Editors will need `imageMagick` and `ghostscript` if vector graphics is to be processed into HTML.

On Windows, it is recommended to run IVOATEX within cygwin, where all dependencies can easily be installed from cygwin's repository.

On Debian-derived systems, the dependencies should be present after running a distribution-specific adaption of

```
apt install build-essential texlive-latex-extra zip xsltproc\
  texlive-bibtex-extra imagemagick ghostscript cm-super librsvg2-bin
```

(cm-super contains vector versions of computer modern fonts in T1 encoding), on RPM-based systems something like

```
yum install texlive-scheme-full libxslt make gcc zip\
  ImageMagick ghostscript
```

should pull in everything that is necessary.

With OS X, a convenient way to obtain the dependencies is to install MacPorts[3] and then run

```
port install ImageMagick libxslt ghostscript +full
```

The canonical OS X TEX distribution is the MacTEX version of TEXLive[4]. It is also possible to build TEX using MacPorts (with `port install texlive`), but this may result in a slightly non-standard distribution.[5]

To see if the full prerequisites are there and compatible with IVOATEX, try building an updated version of this document from its github source (normal users can go without making HTML and packages, in which case a lot of dependencies are not needed):

```
git clone --recurse-submodules https://github.com/ivoa-std/ivoatexDoc
cd ivoatexDoc
make biblio  # update the bibliography
make forcetex # make a PDF ignoring timestamps
make ivoatexDoc.html # make an html document
make package # make a zipfile for IVOA submission
```

During HTML generation, various diagnostics both from tth and from xsltproc (unknown commands, unexpected end tags, and the like) are expected at this point and no reason for alarm; we work on reducing the amount of spurious error messages.

2.2 Basic IvoaTEX operation

For ease of installation and robustness, IVOATEX for now is designed to be used from within a subdirectory of the directory containing the document

[3] https://www.macports.org/

[4] https://www.tug.org/mactex/

[5] See http://tex.stackexchange.com/questions/97183/. Also, MacPorts does add `texlive` as a dependency on many packages, and so frequently *insists* on trying to build it; if you want to prevent this, there is some discussion at http://comments.gmane.org/gmane.os.apple.macports.user/21526.

sources (rather than being installed globally). Given that it is fairly compact, having one copy per document seems acceptable.

So, the first step to use IVOATEX is to create a development directory:

```
export DOCNAME=SampleDoc
# this would be your document's short name, e.g., RegTAP, SIAv2)
mkdir $DOCNAME
```

The DOCNAME – which will turn up in URLs, standard identifiers, and the like – should be chosen to be both succinct and expressive, and it should not contain non-alphanumeric characters (the examples given here assume that, too). A name like `SimpleDALRegExt` probably marks the upper limit in terms of length.

While it is clearly preferable if authors use IVOA's designated common version control system[6] from the outset of document development, it is possible to build ivoatex documents locally as well.

2.2.1 Installation from Archive (without version control)

Without version control, it is sufficient to obtain IVOATEX from a distribution site and unpack it into the future document directory:

```
cd $DOCNAME
curl http://ivoatex.g-vo.org/ivoatex-latest.tar.gz \
  | tar -xvzf -k
sh ivoatex/make-templates.sh $DOCNAME
```

The shell script will print some error messages because no version control has been enabled. These are safe to ignore.

2.2.2 Installation with git version control

The recommended way to run ivoatex is to use git's submodule feature. This makes it very simple to keep ivoatex up to date without polluting the document's history, and it makes it straightforward to feed back any improvements you may make to ivoatex. Hence, you will usually start a document like this:

```
cd $DOCNAME
git init
git submodule add https://github.com/ivoa-std/ivoatex
sh ivoatex/make-templates.sh $DOCNAME
git commit -m "Starting $DOCNAME"
```

The `ivoatex/startup.sh` script will copy a few template files from the ivoatex distribution to the working space. You could do that manually, too; see the script for what files to move where.

[6] Which, despite many concerns (Zuboff, 2019), currently is github.com.

At this point, create a repository in your account using github.com's web interface. Use $DOCNAME as the repository name, and do not choose any template; this procedure will leave you at a web page with a URI like `https://github.com/msdemlei/ivoatexDoc`. On that page, you can obtain a "Clone URI", which is what you can push to. Use the ssh variant, i.e., something like `git@github.com:username/docname.git`.

Then, push your newly changed material into this new repository's main branch:

```
git remote add origin <Clone URI>
git push --set-upstream origin main
```

(depending on your git version, you may have to use "master" rather than "main").

2.2.3 Beginning the document

Main metadata in the Makefile For convenience, the document production should start from some common templates which are part of the IVOATEX distribution. The above procedure has already created `README.md` (a brief introduction to what the document is about), `$DOCNAME.tex` (the future document) and `Makefile` (where some of the document metadata is defined).

The next step is to fill out the makefile template. As of the formatting of this document, this template looks like this:

```
# ivoatex Makefile.  The http://ivoa.net/documents/notes/IVOATex
# for the targets available.

# short name of your document (edit $DOCNAME.tex; would be like RegTAP)
DOCNAME = ????

# count up; you probably do not want to bother with versions <1.0
DOCVERSION = 1.0

# Publication date, ISO format; update manually for "releases"
DOCDATE = ???

# What is it you're writing: NOTE, WD, PR, REC, PEN, or EN
DOCTYPE = ???

# An e-mail address of the person doing the submission to the document
# repository (can be empty until a make upload is being made)
AUTHOR_EMAIL=???

# Source files for the TeX document (but the main file must always
# be called $(DOCNAME).tex
SOURCES = $(DOCNAME).tex

# List of image files to be included in submitted package (anything that
# can be rendered directly by common web browsers)
FIGURES =
```

```
# List of PDF figures (figures that must be converted to pixel images to
# work in web browsers).
VECTORFIGURES =

# Additional files to distribute (e.g., CSS, schema files, examples...)
AUX_FILES =

-include ivoatex/Makefile

ivoatex/Makefile:
	@echo "*** ivoatex submodule not found. Initialising submodules."
	@echo
	git submodule update --init

test:
	@echo "No tests defined yet"
```

All lines with question marks must be filled out. The document date is the publication date, which can be significantly different from the current date. After its initial setting, it should only be changed at the time of submission to the document archive. It is always in DALI-style ISO format (Dowler and Demleitner et al., 2017), e.g., 2014-03-31.

`SOURCES` is used in dependency processing. It would be amended when the source file is split into separate files or if material is included into the document, e.g., via `lstinputlisting`. Graphics files included do not need to be given here, as the document will automatically depend on them. The exception is `role_diagram.pdf`; see sect. 2.2.3.

`FIGURES` must contain the names of all bitmap graphics included in the document; files missing here will be missing from the package for distribution to the IVOA document repository, which will break the HTML rendering. As to `VECTORFIGURES`, see sect. 3.4.2.

`AUX_FILES` is intended for files that should be included in the upload to the IVOA document repository while not taking part in the actual formatting. This in particular concerns XML Schema files, for which the IVOA maintains a separate repository, but is by no means limited to them.

Additional metadata in the LaTeX source The template for the TEX source contains several lines with multiple question marks. These must be filled out as well.

As illustrated in the template, both `author` and `previousversion` support an optional argument giving an URL; for `author`, it should normally point to the respective person's page in the IVOA wiki, for `previousversion`, it should point to the landing page of the respective document version in the IVOA document repository. Further automation for maintaining document history certainly is desirable, and the authors welcome ideas for how this might look like.

`ivoagroup` should contain the name of the (one) IVOA working or interest group under which auspices the document development mainly happens. If this is an interest group (which is only possible for notes), use `ivoagroup[IG]{...}` to make IVOATEX use "Interest Group" rather than "Working Group" as the label of the corresponding piece of metadata.

Pick one from the following list[7]:

- Applications
- DAL
- Data Access Layer
- Data Models
- Grid and Web Services
- Registry
- Data Curation and Preservation
- Standards and Processes
- Semantics
- Operations
- Radio
- Theory
- VO Event
- Time Domain
- Education
- No Group (only possible for Notes)

The README The `README.md` template should be edited to give a short (one-paragraph) introduction to what the document is about at an even higher level than the abstract. The template also contains instructions for building, pointers to the IVOA document repository, and a license statement. While you can change these if you want, that should not usually be necessary.

The Architecture Diagram An architecture diagram is only necessary for documents on the recommendation track. Notes may have one, but since Notes are not shown on it, that is rather unusual.

To prepare an architecture diagram, first obtain the full description of all IVOA standards:

```
cp ivoatex/archdiag-full.xml role_diagram.xml
git add role_diagram.xml
```

This is just an abstract specification that is compiled into images. To ensure this happens, `role_diagram.svg` to `FIGURES` in the Makefile. As long as the LATEX toolchain does not support the SVG image format, you must additionally append `role_diagram.pdf` to `SOURCES`.

[7] We do not enforce the use of the controlled vocabulary. Automatic submission with `make upload` will fail if you invent something here, though.

Then edit `role_diagram.xml` and remove all references to standards unrelated to the current document. As a rule, all `rec` elements for standards not mentioned in "Role within the IVOA Architecture" should be removed. Finally, use a *`thisrec`* element for the current standard. An architecture diagram for VOResource would thus be specified like this:

```
<archdiag xmlns="http://ivoa.net/archdiag">
  <thisrec name="VOResource" x="55" y="155"/>
  <rec name="RegTAP" x="55" y="180"/>
  <rec name="RegistryInterface" x="55" y="205"/>
  <!-- and a few more -->
</archdiag>
```

The standard-specific architecture diagram will be built by saying `make role_diagram.svg` and can be viewed using, for instance, common web browsers. Please refrain from editing the SVG with vector graphics programs. In the PDF renderings of the document, a PDF version of the architecture diagram is required. This is built using `make role_diagram.pdf`, and a stand-in will be built if your system lacks the software to do this conversion (currently: rsvg-convert).

As long as the tooling situation regarding LaTeX and SVG is as unsatisfactory as it is, please commit both `role_diagram.svg` and `role_diagram.pdf` to the version control system.

Source code conventions IvoaTeX requires the source to be written in UTF-8 encoding, since the references shipped with it are in UTF-8. Authors are urged to keep lines shorter than 72 characters in input files whenever possible in order to keep diffs useful and readable. When edits are made, paragraphs should not normally be reflowed to avoid large diffs for minor edits. Authors desiring a reflow after many edits are encouraged to concentrate them in a separate, reflow-only commit.

Makefile targets The PDF version of the document is built by the makefile's default rule, so running `make` will usually by enough. This will produce a file `$DOCNAME.pdf`.

Other makefile targets for author use include:

- `biblio` updates the bibliography (i.e., runs BibTeX); running this is necessary after one of the bibliography files is updated or when a new publication is referenced from the document. This target is not necessary on machines that have latexmk.
- `forcetex` rebuilds the PDF unconditionally just as if the main source had been edited. This target is probably not useful on machines with latexmk.

Note

Simply running `latex`, `pdflatex`, or `mklatex` directly (rather than through make) is *not* supported with IVOATEX. Due to the non-global installation of the support files, the TEX run needs a special environment that is prepared by the makefile.
Also note that on systems without latexmk, bibliography processing must be initated manually by running `make biblio`; unless authors checked in the bbl file this produces, this must be run after a document checkout.

- `$DOCNAME.html` generates an HTML rendering of the document; at this point, this will typically emit quite a bit of spurious diagnostics. Unfortunately, real problems may hide within. Therefore, we currently recommend a visual inspection of the resulting HTML before submission.

- `package` generates a zip file containing everything needed for publication in the IVOA's document repository. Obviously, `DOCVERSION`, `DOCTYPE`, and `DOCDATE` in the Makefile should be updated as necessary before this target is built. The result is a zip file with a name compliant to the IVOA document standards (Genova and Arviset et al., 2017).

- `arxiv-upload` produces a file arxiv-upload.tar.gz for the current document. ArXiv uploads are automatically done by the document coordinator for RECs and ENs, and editors *should not* do them themselves. Authors of Notes may upload their documents to arXiv.

- `upload` produces a package, shows metadata to be uploaded to the IVOA document repository (make sure everything is as you expect it) and then uploads the package.

- `generate` is used to update machine-generated content in the document, typically by the main editor. See section 3.9 for details.

- `update` updates the ivoatex submodule to whatever is in the main ivoatex branch on github. This will fail if you have uncommitted local changes to your ivoatex.

- `$DOCNAME-draft.pdf` generates a PDF rendering just like a simple make, but it will add a watermark against distribution. This is mainly intended for continuous integration workflows. It only works if `pdftk` is installed.

2.3 Examples

Examples for IVOATEX use are found in the ivoa-std organisation on github[8].

3 Authoring documents

While IVOATEX documents can be written much like any other TEX document, it is advisable to follow certain standards and use special facilities for common appearance, easier development, and possible evolution of IVOATEX itself.

3.1 IvoaTEX Features

IVOATEX provides a small set of macros and environments designed to ease standards authoring. These include:

`author`, `previousversion`, `ivoagroup`
: these are discussed in sect. 2.2.3.

`ucd`
: a macro for marking up UCDs. This in particular helps with hyphenation of these, typically long, strings.

`xmlel`
: a macro for marking up XML element or attribute names and similar.

`vorent`
: for a name taken from VOResource or its extensions, usually an element or attribute name.

`admonition`
: This is an environment for displayed boxes, intended for notes, tips, and the like. It takes an argument giving the head of the box, e.g.,

```
\begin{admonition}{Note}
Admonitions should not be overdone.
Also, they are floating insertions.
\end{admonition}
```

`bigdescription`
: This is an environment for definition lists in the style of HTML *dl*, and it will be translated into one. Use `\item[term]` for the term to be defined (the construct this item is in is a `bigdescription`).

`auxilaryurl`
: A macro expanding a path relative to the current document to a full URL. This is discussed more extensively in sect. 3.6

[8] https://github.com/ivoa-std

`inlinetable`
: This is an environment for showing tables in-place (rather than as an insertion potentially somewhere else, as LaTeX's table environment does). This may lead to rather empty pages when the tables get longer.

The intention behind macros like `xmlel` and `vorent` is that such terms are typeset uniformly across documents. Further semantic markup like this is planned for future releases, and document authors are encouraged to contribute terms.

Also note that the title page is generated by the abstract environment. Thus, all IVOATEX documents must have an abstract within the `abstract` environment.

3.2 Listings, Verbatim Material

IVOATEX documents should use the `listings` package to include source code snippets, XML fragments and the like. While the ivoa class requires the package as of version 1.1 (and hence you can use the `lstlisting` environment without an explicit `usepackage` declaration), you will usually want to configure the package in the document preamble (i.e., before its `\begin{document}`), perhaps like this:

```
\lstloadlanguages{XML,sh}
\lstset{flexiblecolumns=true,tagstyle=\ttfamily,
  showstringspaces=False}
```

Additional languages supported that are likely relevant in a VO context include C, fortran, python, SQL, and java, specified as above case-insensitively.

The setup in the example (flexible columns, tags in typewriter, no explicit blanks even in strings) produces reasonably pleasant output for a wide range of languages.

Actual listings are obtained with code like

```
\begin{lstlisting}[language=XML]
<example id="empty"/>
\end{lstlisting}
```

Alternatively, entire files can be included like this:

```
\lstinputlisting[language=XSLT]{makeutypes.xslt}
```

In the PDF rendering, the listings are pretty-printed. In the HTML rendering, the content is, currently, simply included in *`pre`* elements.

If a more compact rendering of listings is desired, for instance, because larger portions of source code are required in the document, listings'

basicstyle option should be used together with one of LaTeX standard size macros. This could be in the argument of lstset in the preamble for a global setting, or on a case-by-case basis as in

```
\begin{lstlisting}[language=tex,basicstyle=\footnotesize]
Other sizes include \tiny, \scriptsize, \small, \normalsize, \large
\end{lstlisting}
```

As of version 1.2 of IVOATEX, only footnotesize is actually formatted by the CSS embedded in the HTML document, and we believe listings should not be much smaller than that anyway. As the options are translated into CSS classes, it is fairly easy to add further formatting functionality on a document-by-document basis, though.

3.3 References and Bibliography

3.3.1 Built-in Bibliographies

IVOATEX documents should use natbib and BIBTEX to manage references. The package comes with two default bibliographies:

- ivoatex/ivoabib.bib containing records for many publications likely to be cited by IVOA documents. Feel free to add further records if they might reasonably be cited by other IVOA documents.
- ivoatex/docrepo.bib containing records for IVOA documents listed in ADS, i.e., recommendations, endorsed notes, and a selection of other notes.

Without latexmk, changes to the document that introduce new references or changes to the bibliography require that make biblio is run before changes become visible.

IVOATEX comes with a bibliography style of its own, derived from agsm.bst. The custom bibliography style was derived to optimise some types of sources uncommon outside of the VO community, in particular IVOA recommendations and notes. Users are welcome to improve ivoatex/ivoa.bst.

3.3.2 Citation Style

As usual in natbib, actual references are made through either writing \citep{tag}, yielding a form like "(Einstein 1905)", or \citet{tag}, yielding a form like "Einstein (1905)". IVOATEX does not support variant forms of citep and citet (i.e., those with optional arguments) yet; they will work in PDF output but fail in HTML. Contributions to improve this are welcome.

To reference an IVOA recommendation, locate its bibcode either using ADS or directly within ivoatex/docrepo.bib and then use one of the cite

macros. The preferred style is to introduce a short name for the standard once with a citation and then use that short name in the remainder of the document to have expressive texts not overciting. For instance,

```
IVOA Identifiers \citep{2016ivoa.spec.0523D} introduces URIs to
reference Registry records, which are typically transmitted in
VOResource \citep{2018ivoa.spec.0625P} format. Both VOResource
and IVOA Identifiers are based on various W3C standards.
```

will come out as

> IVOA Identifiers (Demleitner and Plante et al., 2016) introduces URIs to reference Registry records, which are typically transmitted in VOResource (Plante and Demleitner et al., 2018) format. Both VOResource and IVOA Identifiers are based on W3C standards.

Recommendations should generally be cited in the last version available at the time of writing[9]. If works in progress (working drafts, proposed recommendations are to be cited, authors should create a local bibliography (see below).

As part of bibliography management, authors should occasonally run

```
make bib-suggestions
```

This will suggest updates to what IVOA standards the document cites based on which documents obsoleted which other documents. Note that these suggestions should not be followed where references actually are version-sharp.

3.3.3 Local Bibliographies

When a reference is really only relevant to a single document or is conceptually non-permanent – this, in particular, pertains to Working Drafts or Proposed Recommendations to be cited –, it should be kept in a local bibliography. To instruct LaTeX to use the references there, the declaration at the foot of the document needs to be changed to

```
\bibliography{ivoatex/ivoabib,ivoatex/docrepo,localrefs}
```

Then a file `localrefs.bib` is created and checked into the version control repository. An example BibTeX record for an in-progress document is

```
@Misc{wd:Datalink,
  author={Patrick Dowler and Francois Bonnarel and
```

[9] If a published recommendation is missing in the bibliography, this can be fixed by cd-ing into the ivoatex folder and saying `make docrepo.bib`.

```
    Laurent Michel and Tom Donaldson and David Languignon},
  editor={Patrick Dowler},
  howpublished={{IVOA Working Draft 22 October 2013}},
  title={DataLink},
  year=2013,
  url={http://ivoa.net/documents/DataLink/20131022/}
}
```

Note that *howpublished* contains the precise document date and the *url* points to the actual versioned landing page, not the generic one for the standard.

3.4 Graphics

3.4.1 Bitmap Graphics

IVOATEX supports all bitmap graphics formats that pdflatex supports. In practice, authors are encouraged to restrict themselves to JPEG, PNG, and possibly GIF. Currently, identical images are used for both PDF and HTML renderings. The recommended pattern for figures is

```
\begin{figure}[th]
\begin{center}
\includegraphics[width=0.9\textwidth]{mydiagram.png}
\end{center}
\caption{A diagram of what this is about.}
\label{fig:mydiag}
\end{figure}
```

This gives LATEX some leeway in placing the figure, defines the image size in units of the page width, and centres the image itself.

All bitmap graphics in a document must be listed in the makefile's `FIGURES` variable. If they are not, the HTML rendering will be broken.

3.4.2 Vector Graphics

The only vector graphics format well supported in IVOATEX is PDF. PDF files can be directly used in `includegraphics`. The names of such figures must be listed in the makefile's `VECTORFIGURES` variable.

From `VECTORFIGURES`, IVOATEX arranges that, when a PDF figure `foo.pdf` is used, the HTML target depends on a file called `foo.png`. This PNG can be generated automatically by IVOATEX using a combination of ghostscript and ImageMagick. It may sometimes be preferable to perform a custom conversion by hand (e.g., more compact representation with bilevel source images), in which case the pre-rendered PNG should be included in

the version controlled repository. This also has the advantage that neither ghostscript nor ImageMagick are build dependencies of the document.

3.5 Tables

In tables, rules should be used sparingly. The standard pattern for tables is something like

```
\begin{table}[th]
\begin{tabular}{p{0.35\textwidth}p{0.64\textwidth}}
\sptablerule
\textbf{Column Head}&\textbf{Another column head}\\
\sptablerule
A value & Another value\\
A value in row 2& And so on\\
\sptablerule
\caption{A sample table}
\label{table:extable}
\end{tabular}
\end{table}
```

The `sptablerule` used here inserts a horizontal rule with some extra spacing and will be rendered consistently in both PDF and HTML. It should not, as a rule, used between table rows. It is intended primarily to delimit the table itself as well as the the heading and the body.

3.6 Hyperlinks

While IVOATEX puts no restrictions on the usage of hyperref features, the preferred way to include links in IVOATEX documents is to use the `url` macro, i.e., use the URL itself as the anchor text. In this way, the link remains (to some extent) usable even if the document is printed. The alternative two-argument `href` should generally be avoided as it fails on paper. For instance,

```
(this is bad:) The \href{http://ivoa.net}{IVOA} has issued
\href{http://ivoa.net/documents}{many standards}.
```

would severely degrade when printed and is hence discouraged, whereas

```
The IVOA\footnote{\url{http://ivoa.net}} has issued many
standards, all of which can be retrieved from
\url{http://ivoa.net/documents}.
```

works properly on all of IVOATEX's target media.

With non-HTTP URIs, it is recommended to use hyperref's `nolinkurl` macro (rather than an unadorned `texttt` or similar); advantages include that line breaking is better with nolinkurl, less manual escaping is necessary, and, if desired, such URIs can be more easily styled. An example:

```
The value of the \xmlel{standardID} attribute will be
be \nolinkurl{ivo://ivoa.net/std/Registry#OAI-2.0}.
```

A special feature related to links is the `auxiliaryurl` macro. It is used when a document references a resource coming with the document and versioned with it, but not included with the document text. Examples for such material could be helper scripts, simple validators, larger bits of sample data, and the like. Note that vocabularies and XML schema files should not be distributed in this way, as they have conventional places within the ivoa.net URL hierarchy.

For instance, `\auxiliaryurl{custom.css}` expands to

https://www.ivoa.net/documents/ivoatexDoc/20220614/custom.css,

in this document, which is, for each release of this document, the URL the example custom CSS file mentioned in the example in sect. 5.4 is found at. The preferred way to include such (long) URLs is in footnotes, as they typcially cannot be hyphenated.

In order to make IVOATEX include the file linked to in this way in the package submitted to the document repository, you must add the argument of `auxiliaryurl` to `AUX_FILES` in the Makefile.

3.7 Editorial tools

When authoring standards, it is sometimes necessary to include editorial comments of the type "Need to clarify" or "Specification incomplete". We recommend to use the todonotes package for such pieces of text[10]. The recommended usage is like

```
...
\usepackage{todonotes}
...
\begin{document}
\todo{This is an example for a editorial note}.
```

This is an example for a editorial note

A rendering of such a to-do note is shown in this paragraph. In HTML output, only the simple `todo` macro without options is supported, and text is simply displayed inline right now. Note in particular that todonote's `obeyDraft` and `obeyFinal` package options are ignored. Although IVOATEX does not enforce it (yet), finished recommendations should have no todo items in them.

While todonotes is a useful tool for standards development, we discourage the use of packages to mark up changes, as maintaining such markup usually is very hard, and version control offers a more manageable solution to the problem such packages attempt to solve.

[10] Full documentation is available at `http://www.tex.ac.uk/CTAN/macros/latex/contrib/todonotes/todonotes.pdf`.

3.8 Version Control System Information

It is recommended to include basic metadata obtained from the version control system into IVOATEX Documents where available. Basic support for subversion and git is built into IVOATEX, based on some sort of keyword substitution. The underlying mechanisms, as well as the information conveyed, is rather different between git and subversion. We cover the git case first; the part about how this was done in subversion is probably only of historical interest now.

3.8.1 VCS Metadata in Git

To enable in-document VCS tracking in git, two steps are necessary:

1. Add `gitmeta.tex` to your `SOURCES` in the makefile.
2. Add a line `\input gitmeta` somewhere near the top of your LaTeX source (right below `\input tthdefs` is a good place).

This will then add the commit identifier and a time stamp to the title page of the document. Patches to make ivoatex include some sensible URL (perhaps the remote origin URL if it exists) are solicited.

3.9 Generated Content

Sometimes it is desirable to have parts of a document generated through some sort of ivoatex-external process. Examples include copying documentation from XML Schema files into the standard document or obtaining column metadata for standard data models from a live TAP_SCHEMA.

For such cases, IVOATEX offers a python script `update_generated.py`, which is executed by the `generate` make target. It simply looks for structured comments in the main document and replaces what is between them with generated contents. The opening line consists of a TEX comment introducer, a blank, the literal `GENERATED:`, another blank, and a command line. The closing line is a comment consisting of `/GENERATED`.

On running `make generate`, the material between the opening and the closing line is replaced by the output of the command.

For instance, in the following snippet the material between the comments was inserted by `make generate`:

% GENERATED: echo This is generated content
`This is generated content`

% /GENERATED

`make generate` will not replace any content in the document source if even just one command fails to execute as indicated by the command's return code; it is transactional in this sense.

In addition to allowing arbitrary shell commands, `update_generated.py` has a facility that allows calling special python functions from within documents: Commands starting with an exclamation mark ("!") are translated to calls to appropriately named python functions defined within `update_generated.py`.

Currently, there are two such builtin commands; both are somewhat experimental features that may change when more experience has been gathered as to their usefulness.

One command is `taptable`, which extracts documentation from a live `TAP_SCHEMA`. The access URL of the live TAP service must be specified in an environment variable TAPURL in the Makefile, for instance

```
TAPURL=http://dc.g-vo.org/tap
```

Then, in the document one can use the structured comment

```
% GENERATED: !taptable tap_schema.tables
% /GENERATED
```

After a `make generate`, the LaTeX source for a table describing the columns of `tap_schema.tables` will be between the two comment lines; rerunning `make generate` will replace that content with a refreshed version.

The second command implemented is `schemadoc`. This formats documentation for a type in an XML schema file. This only works properly if the XML schema defines a *`vm:targetPrefix`* element in the way pioneered by Ray Plante in VOResource.[11] With an appropriately instrumented schema, a document can say

```
% GENERATED: !schemadoc SchemaSource.xsd MyType
% /GENERATED
```

to produce documentation for the schema type *`MyType`* within the XML schema file `SchemaSource.xsd`.

IvoaTeX will never execute `update_generated.py` as part of a dependency chain; it is intended that `make generate` must always been manually triggered. On the one hand, this is because its dependencies cannot be generically modelled, given that arbitrary commands can be executed. Document authors are also discouraged from providing such dependency information – it is fairly common that content generation depends on the availability of external resources (e.g., databases or network services), and a document build should not fail just because these are unavailable.

[11] See https://github.com/ivoa-std/TAPRegExt for a document using schemadoc.

We mention in passing that generated content puts potentially executable material into documents, which is of course an attack vector for malicious software. However, calling `make generate` is no additional security risk, as whoever can change the document can probably change the makefile, too, and the makefile can already contain arbitrary commands that will be executed on the calling user's behalf.

3.10 The Standards Record

IVOA recommendations and endorsed notes must be registered using the schema from StandardsRegExt (Harrison and Burke et al., 2012). A major use case for this is to allow references to standards from other registry records. The prototypcial example for this is the *standardID* attribute of VOResource *capability* elements that contains identifiers (and possibly standard keys as fragment identifiers) declaring "this service works as specified by that standard".

Registration of IVOA-approved standards happens through the registry of registries (RofR, authority ivoa.net). The preparation of the registry records, however, is up to the document editor. It is strongly recommended to keep and maintain the registry record in version control alongside with the document source.

To do that, create a standards record from IVOATEX's template. In a clone of a repository using IVOATEX, you could say:

```
$ cp ivoatex/stdrec-template.xml $DOCNAME.vor
$ git add $DOCNAME.vor
```

In the resulting file, the items one has to change are marked with four hash marks; the elements *schema* and *key* can be removed or repeated as required by a particular standard, all other elements present should be given at least once. Please remove the explanatory comments as you go.

This document comes with a filled-out sample for itself[12]. This is not to be uploaded into the Registry, as Notes are not (normally) registered.

Most of the metadata to be filled in actually already is available in structured form in IVOATEX. Contributions that help automating creation and maintenance of the standards record exploiting this are welcome.

With the upload of a proposed recommendation to the document repository, the editor should send the standards record to the registry of registries by e-mail[13] and then re-submit the record with each upload or a Recommendation or Endorsed Note; interim document states (WD, PR, PEN) are not tracked by the Registry.

[12] https://www.ivoa.net/documents/ivoatexDoc/20220614/ivoatexDoc.vor
[13] See `http://rofr.ivoa.net/` for contact information.

3.11 Adding tests

Many IVOA documents contain machine-readable artefacts or make machine-verifiable claims. If this is true for your document, consider adding a `test` target (since 2022, IVOATEX's template Makefile alrady provides one).

The `test` recipe should be designed so that

- the recipe fails if an assertion fails,
- minimal (ideally no) output is produced for succeeding assertions,
- extra (i.e., non-IVOATEX) software required to run the tests is documented in a Makefile comment right above the recipe.

In the following, we discuss some typical scenarios for which tests should be provided.

3.11.1 Schema Validation

Standards that come with XML schemas should validate both the schema and one or more instance documents, which ideally should exercise as many aspects of the schema as possible. The instance files do not need to be part of the released standard, although the auxilaryurl mechanism (see sect. 3.6) would give a simple way to include these, too, without overly impacting the document's appearance.

In order to limit the variety of tools editors may need, it is recommended to use the `xsdvalidate` subcommand of STILTS (Taylor, 2006) for validation; this subcommand is available only in STILTS versions 3.4-4 or later[14]. The VOResource standard[15] shows an example (edited here for clarity):

```
STILTS ?= java -jar stilts.jar

# These tests need STILTS >=3.4-4
test:
  @$(STILTS) xsdvalidate VOResource-v1.1.xsd
  @$(STILTS) xsdvalidate \
    schemaloc='http://www.ivoa.net/xml/VOResource/v1.0=VOResource-v1.1.xsd' \
    example-voresource.xml
```

Some brief explanations are in order:

- Using a variable for STILTS allows one to easily override the execution path if necessary (e.g., because of out-of-date system packages or custom paths) like this:

[14] The STILTS jar file can be downloaded from http://www.starlink.ac.uk/stilts/stilts.jar

[15] https://github.com/ivoa-std/VOResource

```
STILTS=stilts-beta make test
```

- The comment specifies the dependencies to save later editors or authors trial and error. This becomes particularly important as test suites grow.
- The first line of the recipe validates the schema itself. The leading `@` suppresses output, which reduces clutter in case diagnostics are produced.
- The second line of the recipe validates an instance document. The important thing here is to override the schema for the standard's target namespace with the local copy of the schema; without the extra `schemaloc` parameter, STILTS would validate against the schema from the document repository, which would both hide problems in the new schema and lead to false positives in the instance document validated (as it will not validate against the previous schema if it exercises new features, which it should). Further `schemaloc` parameters, pointing to either local files or network resources, can be used if other schema locations need to be specified. As in the example, use continuation lines (with backslashes) to keep the command lines readable.

3.11.2 Running Queries

Where documents contain sample queries or requests, it is good practice to automatically extract and execute these, ideally with some basic checks for sane results. An example for how this can be organised can be found with the RegTAP standard[16].

To control the number of different tools editors might, it is recommended to base such code on either STILTS or pyVO (Graham and Plante et al., 2014).

In RegTAP, which uses pyVO, the test rule looks like this (edited here for clarity):

```
# These tests require python3 and pyvo (Debian: python3-pyvo)
test:
  @TAP_ACCESS_URL=http://dc.g-vo.org/tap python3 check_examples.py
```

The script `check_examples.py` is too long to be reproduced in full, but its core is:

```
def main():
  svc = pyvo.dal.TAPService(TAP_ACCESS_URL)
  with open("RegTAP.tex") as f:
    for title, sample_query in iter_examples(f):
      try:
```

[16] https://github.com/ivoa-std/RegTAP

```
    result = svc.run_sync(sample_query).to_table()
    if not len(result):
      print(f"WARNING: Example returned no records in sect. '{title}'")
  except Exception as ex:
    print(f"ERROR: Example went bad in sect. {title}")
    print(ex)
    sys.exit(1)
```

Some notes on this:

- Again, suppress echoing of the commands make executes by prefixing recipe lines with `@` in order to keep the output tidy.
- Tests of this kind will almost always use remote network resources. If possible, do not hide these within the (potentially complex) Python code but make them explicit (and easily adaptable) in the Makefile.
- The Python fragment above should essentially do for any standard discussing TAP queries. Note that the script distinguishes between venial (empty results just give a warning and do not abort the test run) and mortal (execution errors make the test run abort immediately; the call to `sys.exit` is important to make the entire make rule fail) sins. It again will produce no output if everything runs fine.
- The `iter_examples` generator used in the Python code is expected to yield pairs of section titles and sample queries. The code in RegTAP uses heuristics strictly valid only for this document (e.g., queries in lstlisting environments, subsections as the relevant sectioning element). The basic idea – have some magic comment to signal that an example is executable – should be widely applicable, though. This way, a verifiable query in the instance document looks like this:

```
%CHECK_HERE
\begin{lstlisting}[language=SQL,flexiblecolumns=true]
SELECT ivoid
FROM rr.resource
 ...
```

3.11.3 Miscellaneous Tests

Tests can also come as simple shell commands (returning non-zero on failure and emitting some diagnostics only in that case) as well. Where these go beyond one-liners, one should probably use a shell script. Any dependency beyond the unix core tools should again be documented in a Makefile comment right above the test rule.

An example for this type of check is again given in the VOResource standard[17], where the test rule then would be:

[17] https://github.com/ivoa-std/VOResource

```
# These tests require xmlstarlet
test:
  @sh test-assertions.sh
```

The shell script called could run multiple tests, in particular if they share some common infrastructure. It is recommended to isolate each test in a shell function with a name starting with "assert" and then calling these shell functions as appopriate. In the VOResource example, where the authors make sure that if a Recommendation is being published, the schema version tag actually is "clean", such a shell function takes a shape like:

```
assert_schema_version_for_rec() {
  # make sure the schema version has no tags if this is a REC
  if xmlstarlet sel -T -t -m "xs:schema" -v "@version" $SCHEMA \
    | grep " ^[0-9]\.[0-9] $" > /dev/null ; then
    : # all is fine, no output
  else
    die "$SCHEMA version has tags (inappropriate for a REC release)"
  fi
}
```

Note the STILTS package mentioned above includes some other tools that may be useful for testing, for instance the subcommands `votlint` for checking VOTables and `datalinklint` for checking DataLink tables; see the documentation[18] for details. In particular it is a good idea to validate all example VOTable documents included in document text.

3.12 Submitting a Document

Snapshots of documents should be regularly submitted to the document repository as part of the IVOA review process. While IvoaTeX could still make this a bit more streamlined, there is some automation for this process present. Consider the following recommendation for the release process (of course, replace `${DOCNAME}` with the DOCNAME as defined in the Makefile):

1. Make sure your ivoatex is up to date; in git checkouts, run `make update` (or, equivalently, `git submodule update --remote`).

2. If the document contains generated content, run `make generate` and inspect the subsequent output of `git diff` for anything unexpected.

3. Add the version-specific URL and designation (e.g., "WD-20100801") of the currently published document (i.e., the version you are replacing) in the document header using `previousversion`.

4. Make sure the changelog appendix lists the major changes since the last release of the document; it is probably a good idea to re-scan

[18] http://www.starlink.ac.uk/stilts/sun256/cmdUsage.html

the (generally more extensive) version control history at this point for anything that is missing from the in-document changelog.

5. If you have schema files, vocabularies, or the like, make sure their version identifiers are what you expect (in particular, for recommendations, such version identifier should have no appendices of the kind "-pr1". The `AUX_FILES` variable in your Makefile is a good start for what files to check.

6. For documents that provide a `test` target, run `make test`; this target must at least not error out and ideally not produce any diagnostics.

7. Run `make` and inspect `${DOCNAME}.log`. There should obviously be no undefined citations or internal references. Also, try to avoid overfull hboxes (underfull hboxes are not always avoidable).

8. Have another round of proofreading on `${DOCNAME}.pdf`. Also critically inspect the architecture diagram and the relationships to other standards, in particular whether new versions of those have reached REC in the meantime. `make bib-suggestions` will do that for you in the case of IVOA recommenations.

9. Run `make ${DOCNAME}.html` and briefly review the produced HTML file. Warnings or even error messages in this process are (regrettably) still to be expected, so at least superficial human inspection of the output is necessary.

10. Edit the Makefile and update the DOCVERSION, DOCDATE, and DOCTYPE variables according to the new document release.

11. If uploading a REC or EN, update (or create) the standards record and send it to the RofR (cf. sect. 3.10).

12. Run `make package`. If no major problems become visible,

13. run `git commit -am "Preparing for the release"` (the message should probably be more specific on *which* release) – this will update the source if VCS information is represented as per section 3.8.

14. Run `make upload`. This will ask you to review the document metadata in a python dictionary, let you add a note for the document coordinator and performs the upload.

15. Review the file `docrepo-response.html` created in the document directory; this contains the raw HTML the uploading script got back from the document repository. We do not automatically check for errors yet, so again human attention is required.

4 Developing IVOA Documents

While IVOATEX can of course be used in private repositories, its specific focus is the development of IVOA standards and related documentation. This section discusses practices and policies put in place to ensure transparent and reproducible processes.

4.1 The Organisations ivoa and ivoa-std

As described above, documents will usually start in private repositories. At the latest when documents become become proposed endorsed notes or working drafts, they must move into the ivoa-std organisation. Documents there are subject to certain policies ensuring a controlled and well-documented evolution (cf. sect. 4.3).

For now, requests to move into ivoa-std are processed by the chair of the TCG; they are usually filed by the chair of the working or interest group handling the document.

For non-reviewed IVOA material, there is the organisation ivoa. Community members are welcome to use it for their VO-related projects and in particular Notes. However, there is no requirement to do that.

4.2 Contributing to a Document

IVOA repositories for standards and notes follow many open source projects in using a fork and pull request workflow for contributing updates. This allows open contributions whilst protecting the repository integrity. All contributors, whether document editors or first time contributors, follow this same process. All pull requests need to be approved by a maintainer of the repository.

When setting up your environment for a first time commit, we recommend following the 'triangle' workflow (Fig. 1) to make later contributions easier.

1. Fork the official repository using the 'Fork' button on the upper right of the repository's GitHub web page. This will create a copy of the repository under your user name which you can directly edit. See the GitHub documentation `https://docs.github.com/en/get-started/quickstart/fork-a-repo` for further information.

2. On your local computer, clone the official repository and then add your fork as a remote repository. Note here that we use a read-only clone (https://) from the IVOA repository to avoid the potential for future mistakes.

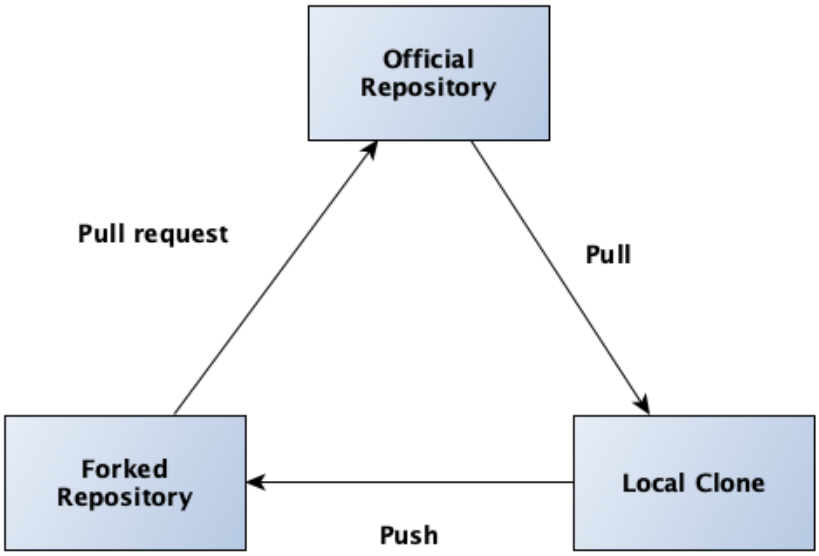

Figure 1: The 'triangle' workflow, where updates are made to a local copy of the official repository, pushed to a fork of the repository and contributed back to the official repository through pull requests.

```
git clone --recurse-submodules https://github.com/ivoa-std/standard.git
cd standard
git remote add mine git@github.com:username/standard.git
git remote -v
```

you should see output similar to

```
mine  git@github.com:username/standard.git (fetch)
mine  git@github.com:username/standard.git (push)
origin git://github.com/ivoa-std/standard.git (fetch)
origin git://github.com/ivoa-std/standard.git (push)
```

3. Now to prepare for your change, make sure you are up to date with the official repository and create a branch for your work.

   ```
   git checkout main
   git pull origin main
   git checkout -b branch-name-for-change
   ```

   IVOA repositories are tranistioning from using "master" as the main branch to using "main". Depending on the repository, you may need to adapt the above commandline accordingly.

4. Once you are happy with the change (and we do recommend building the PDF document locally to review it) you can save it to git and send it to your forked repository. The status command will list everything that has changed and the add command will include it the commit.

   ```
   git status
   git add standard.tex another-changed-file
   git commit -m "a  short and useful description of my changes"
   git push mine branch-name-for-change
   ```

5. Then, on the GitHub web interface for your repository, create your Pull Request to contribute it back to the official repository.

6. Wait for reviews, update the change as needed, repeating step 4 for each set of changes. Once the change is approved by a maintainer, the maintainer may merge your change themselves or leave you to merge once you are ready.

For future changes repeat steps 3 to 6.

4.3 Managing Changes

This section should be read by persons serving as document editors.

The role of a document editor is to ensure that a document eventually converges towards a cohesive text, where an editor may or may not be one of the document authors. Editors should also make sure that document changes are always well-documented and open to later scrutiny. The goal should be that later readers can find out why a given change was made, i.e., what new functionality it was intended to provide or what problem it should address.

While a document is under public review, the editor's role as the text's gatekeeper is particularly important. In particular, they must ensure that all changes are properly reviewed and do not introduce changes so profound that a new full review would be required.

With these considerations, the following policies apply to documents under IVOA management:

a. Nobody can commit directly to the main branch of a document. Instead, all changes go through pull requests, where editors take care that titles and descriptions are suffiently clear and complete.

b. Before public review (i.e., in the state of a Note, a Working Draft, or a Proposed Endorsed Note before TCG review), it is at the editor's discretion whether they want individual pull requests reviewed separately or whether pull requests are merged without a further review. To give some examples, a pull request adding a major feature could

be announced on the pertinent IVOA mailing list and undergo some sort of public pre-review, whereas a re-working of a section could profit from a formal review by a co-author. A pull request just repairing some bibliographic references can be merged by the editor directly.

c. For documents under public review (i.e., Proposed Recommendations and PENs under TCG review), all pull requests must be reviewed, where at least one reviewer should be a document author. It is the editor's duty to find reviewers, where obviously the persons who brought up the points addressed by the pull request would be particularly good choices.

In order to ensure document histories remain readable, editors are encouraged to squash-merge pull requests.

4.4 Adopting a Repository for ivoa-std

This section documents policies and workflows for the managers of the ivoa-std organisation; normal IVOATEX users can ignore it.

On request by a chair of an IVOA working group, the ivoa-std management will migrate a repository from where the original editor started it (called `<original-repo>` below) to the ivoa-std organisation. To do that, they perform the following steps:

1. Inspect the `<original-repo>` and make sure it includes the minimal files: README.md, LICENSE, Makefile, ivoatex submodule, and a LaTeX file with the proposed `<short-name>`
2. Create a new public repository `ivoa-std/<short-name>` via github's browser UI and using the "import" facility to import the `<original-repo>`
3. Make a local clone and build the document; if this fails, create an issue for the original author(s) to fix before any work can proceed
4. Optional: add github workflow, commit, and push directly to main branch
5. Setup branch protection rules for the main branch (see below)
6. Give the WG team and the editor admin privileges (see below)
7. Notify the WG and the editor of the move via the WG's mailing list
8. Notify the original owner, noting that `<original-repo>` should be renamed and preserved for a while, while the repository must now be forked from ivoa-std in order to enable future pull requests (cf. sect. 4.2)

Branch protection rules are set up in the "Settings" tab of the repository (top right). On the left side of the settings page, chose "Branches" and either edit the `main` branch rules (if present) or add a new rule (if not). The standard set of rules that are suitable for most development are:

1. Require a pull request before merging
2. Require approvals
3. Dismiss stale pull request approvals when new commits are pushed
4. Require status checks to pass before merging (see below)
5. Require branches to be up to date before merging

The status checks requirement can be enabled before an actual check (github workflow) is added or found by the system. Repository administrators may have to come back and enable a specific status check workflow after it is added (behaviour is a little opaque here).

By not enabling "Include administrators", repository admins will be able to override the approval requirement and merge pull requests that do not require review (e.g., fixing the build, updating Makefile settings, etc).

Granting admin privileges to the WG (there is a team setup for each WG) is also done in the "Settings" tab. On the left side of the settings page, chose "Collaborators and teams" and then "Add teams" (green button on the right). Start typing the WG name or acronym (e.g. dal or reg) and then select the complete group name from the popup. The WG teams are all of the form `ivoa-std/<group>-admin`; the complete list can be found from the top-level `ivoa-std` organization page in the "Teams" tab.

5 Customisation and Development

This section discusses aspects of IVOATEX that are more technical in nature. Authors with a modicum of TEX expertise are nevertheless encouraged to read it.

5.1 Technical Overview

The central files in IVOATEX processing are

ivoa.cls
: The class file, inheriting from LATEX's article class. The file defines the markup rules for PDF processing, including titlepage generation and extra macros and environments. Its content is ignored for HTML generation.

tthdefs.tex
: This file protects its contents from normal TEX processing by a `\iftth` conditional. This way, only tth sees definitions made here. Each special feature defined in `ivoa.cls` has a counterpart here, giving rules for its translation to HTML. This usually encompasses emitting some HTML before and after the argument of a TeX construct, where material between `\begin{html}` and `\end{html}` is included literally in the HTML document.

tth-ivoa.xslt
: An XSLT stylesheet that postprocesses tth's output and performs some operations that would be inconvenient to implement in `tthdefs.tex`, in particular for the formatting of the opening material.

Makefile
: This makefile is included by the user makefile in the document directory proper. It defines the rules given above as well as some extra housekeeping rules like package building and building tth from its source.

5.2 Semantic Markup

In order to make it support rich, semantic markup, IVOATEX needs to be continuously developed. In particular, it is good practice to define macros for marking up values of certain datatypes, as with IVOATEX's `xmlel` and `vorent`. Thus, whenever a document has multiple instances of such values, authors should define macros and use these. For instance, RegTAP deals with lots of concepts from its own database schema and hence has

```
\definecolor{rtcolor}{rgb}{0.15,0.4,0.3}
\newcommand{\rtent}[1]{\texttt{\color{rtcolor} #1}}
```

in its document preamble to define markup for "RegTAP entity", whereas this note, as it mentions many words with a special meaning to TEX, has

```
\definecolor{texcolor}{rgb}{0.4,0.1,0.1}
\newcommand{\texword}[1]{\texttt{\color{texcolor} #1}}
```

Such macros will be included in IVOATEX itself rather than an individual document's preamble when they prove useful in multiple documents.

5.3 Custom Macros and Environments

The tth translator used by IVOATEX ignores `usepackage`. Many common packages are natively supported, but those that are not in general need specific handling, and sometimes support is somewhat spotty. For instance, the `nolinkurl` macro is not supported natively by tth, and hence in `tthdefs.tex` there is code to the effect of

```
\def\nolinkurl#1{\special{html:<span class="nolinkurl">}%
  \verb|#1|%
  \special{html:</span>}}
```

When a document requires special markup, it is likely that different implementations will be necessary for PDF and HTML output. Using `iftth` the implementations for the current output mode can be selected (without the `newif` mentioned in the tth documentation, as that is already performed in `tthdefs.tex`).

For instance, RegTAP 1.0 had many inline tables that need special spacing for the PDF rendering, whereas normal tables will do for them in HTML. It therefore had in its preamble the definitions

```
\iftth
  \newenvironment{inlinetable}{}{}
\else
  \newenvironment{inlinetable}{\vskip 1ex\vfil
    \penalty8000\vfilneg%
    \hbox to\hsize\bgroup\hss}
  {\hss\egroup\vspace{8pt}}
\fi
```

(this mechanism proved useful for other specifications, too, and so it is part of IVOATEX proper now).

5.4 Custom CSS

If you find you need custom CSS to fix HTML formatting, you should probably talk to IVOATEX's authors first. There are, however, legitimate cases when something needs extra styling in HTML that comes out right without further effort in the PDF output. In such cases, a custom CSS file can be added to a repository (it must then also be added to `SOURCES` in the Makefile in order for it to be delivered with the document package).

The document itself would then use the IVOATEX's `customcss` macro in its preamble with the CSS file name as an argument. For example, the source for this document says

```
\iftth
 \newcommand{\comicstuff}[1]{
    \begin{html}<span class="comic">#1</span>\end{html}}
\else
  \newcommand{\comicstuff}[1]{(HTML exclusive material)}
\fi
```

in its preamble. With this and a CSS file[19],

[19] https://www.ivoa.net/documents/ivoatexDoc/20220614/custom.css

`\comicstuff{If this is comic sans, your web browser is permissive.}`

becomes: (HTML exclusive material)

5.5 Maintenance of the Architecture Diagram

The IVOA architecture diagram is used by Dowler and Evans et al. (2021) to visualise the standards landscape. All IVOA recommendations should have an architecture diagram showing the current standard as well as the related standards within that landscape.

Within IVOATEX, architecture diagrams are produced in Scalable Vector Graphics (SVG). The source is in `ivoatex/archdiag-full.xml`, specifying the location of the recommendations (in *`rec`* elements) and the documents on the recommendation track (in *`prerec`* elements). The figure has a design size of 800×600 "pixels". Note that SVG is not really pixel-based – the numbers are just convenient, unitless floating point numbers, and the conversion of these coordinates to physical ones is done at render time.

In the diagram, the standards shoud be limited to the inner box, i.e., the zone between 50, 100 and 750, 500. Standards boxes are 18 pixels high, which is set in the `format-standard` template in `make-archdiag.xslt`. Their widths default to 90 pixels, but when you add a box in `archdiag-full.xml`, please take a moment to provide a `w` attribute. See the opening comment of `archdiag-full.xml` for a brief howto on having the machine compute it.

When standard boxes are aligned, the vertical distance of their centres should be 25 pixels, the horizontal distance 100 pixels. To keep the diagram lively, standards not obviously grouped with others may be placed "off-grid".

The specifications in `archdiag-full.xml`, as well as those in the authors' `role_diagram.xml`, are ad-hoc XML interpreted by the stylesheet `ivoatex/make-archdiag.xml`. That stylesheet is written such that all three levels of the architecture diagram can be created. The `Makefile` in `ivoatex` has the necessary rules, but in contrast to the author rules, they are expected to be executed *within* the ivoatex directory.

So, to create the full versions of the three levels of the architecture diagram, do something like

```
cd ivoatex
make archdiag-l0.svg
make archdiag-l1.svg
make archdiag-l2.svg
```

The resulting svg files can be viewed in common browsers or in vector graphics programs like inkscape. The latter can also be used to find good positions for new elements (cursor coordinates are shown in the footline, but the direction of the y axis is reversed versus the SVG coordinates). Please do not use graphical tools to edit the diagram itself – the goal of the

current architecture is to make edits transparent and have a clear and simple specification of the standards themselves in a file producing meaningful diffs.

6 Desirable Features to be Implemented

A major drawback of IVOATEX's HTML output is that paragraphs are not actually marked up as such. Due to the TEX processing model, their reconstruction is non-trivial. Hence in the generated HTML, source-level paragraphs are rendered as text nodes separated by empty HTML paragraph elements. It would probably be possible to rectify this in the XSLT postprocessing.

An automated way to maintain the in-document history (i.e., the sequence of `\previousversions` in the preamble) and better support to generate the change log would be desirable.

The biblio and forcetex targets behave in somewhat confusing ways. Perhaps we should simply make latexmk a hard dependency and forget about them?

A Changes from Previous Versions

A.1 Changes from Version 1.2

- Changed instructions to use github rather than volute.
- Added a best practice on contributing via github.
- Added policy sections for editors and the IVOA github maintainer.
- Added info on a few new ivoatexDoc features (e.g., the ucd macro).
- Added a section on writing regression tests.
- Now mentioning bib-suggestions for bibliography maintenance.
- Expanded the submission checklist.
- Dropped a section on ivoadoc migration; it's unlikely anyone will need that any more.

A.2 Changes from Version 1.1

- Added material on the SVG architecture diagram, removed references to old PNG-based workflow.
- Added an upload checklist (and, relatedly, listing legal WG names).

- Added documentation on auxiliaryurl.
- Added instructions on making standards records.
- Documented triple-bibliography maintenance.

A.3 Changes from Version 1.0

- Changed google code URLs to volute.g-vo.org ones.
- Documented new facilities for generating material, with extra focus on auto-documenting XML schema.
- Added advice on citing IVOA recommendations.
- Re-targeting for IVOATEX1.0 (rather than 0.4 before)

References

Demleitner, M., Plante, R., Linde, T., Williams, R. and Noddle, K. (2016), 'IVOA Identifiers Version 2.0', IVOA Recommendation 23 May 2016, arXiv:1605.07501.
http://doi.org/10.5479/ADS/bib/2016ivoa.spec.0523D

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2017), 'Data Access Layer Interface Version 1.1', IVOA Recommendation 17 May 2017.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0517D

Dowler, P., Evans, J., Arviset, C., Gaudet, S. and Technical Coordination Group (2021), 'IVOA Architecture Version 2.0', IVOA Endorsed Note 01 November 2021.
https://ui.adsabs.harvard.edu/abs/2021ivoa.spec.1101D

Genova, F., Arviset, C., Demleitner, M., Glendenning, B., Molinaro, M., Hanisch, R. J. and Rino, B. (2017), 'IVOA Document Standards Version 2.0', IVOA Recommendation 17 May 2017.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0517G

Graham, M., Plante, R., Tody, D. and Fitzpatrick, M. (2014), 'PyVO: Python access to the Virtual Observatory', ascl:1402.004.
https://ui.adsabs.harvard.edu/abs/2014ascl.soft02004G

Harrison, P., Burke, D., Plante, R., Rixon, G., Morris, D. and IVOA Registry Working Group (2012), 'StandardsRegExt: a VOResource Schema Extension for Describing IVOA Standards Version 1.0', IVOA Recommendation 08 May 2012, arXiv:1402.4745.
http://doi.org/10.5479/ADS/bib/2012ivoa.spec.0508H

Plante, R., Demleitner, M., Benson, K., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2018), 'VOResource: an XML Encoding Schema for Resource Metadata Version 1.1', IVOA Recommendation 25 June 2018.
http://doi.org/10.5479/ADS/bib/2018ivoa.spec.0625P

Taylor, M. B. (2006), STILTS - A Package for Command-Line Processing of Tabular Data, *in* C. Gabriel, C. Arviset, D. Ponz and S. Enrique, eds, 'Astronomical Data Analysis Software and Systems XV', Vol. 351 of *Astronomical Society of the Pacific Conference Series*, p. 666.
https://ui.adsabs.harvard.edu/abs/2006ASPC..351..666T

Zuboff, S. (2019), *The age of surveillance capitalism*, Profile Books.