Latex to InDesign

A JavaScript system to convert
Latex texts to InDesign

Revision 6

Contents

Introduction

The aim of the scripts described in this document is to convert Latex texts to InDesign. The extent to which the conversion is successful will vary depending on the state of the Latex file. The conversion may at times prove to be problematic.

Especially complex equations can be a problem. Simple to moderately complex equations are handled fine, but complex equations can present difficulties that the script can't yet cope with, and Latex's equation arrays are not yet supported. The text offers some solutions for equations that the converter could not or not completely handle.

In addition, there are several Latex versions and many packages that extend the functionality of a Latex installation and it's impossible to provide support for all of them. (The InDesign equivalent of a package is a plug-in or a script.)

In some cases, additional commands can be incorporated without any problem. For example, the `booktabs` package defines three rule commands (`\toprule`, `\midrule`, and `\bottomrule`), which are easy to handle. On the other hand, the Ti*k*Z drawing package provides an environment for creating diagrams, which the script can't replicate. Those diagrams should therefore be done manually in InDesign and placed separately. (It's possible to recreate the package and have InDesign create the drawings, but at this stage it's not worth the effort.)

Scripts and videos

The scripts can be downloaded using this link. That web page also has several videos that show several parts of the scripts in action. Some of the videos are a bit out of date because I don't always rerecord them after making changes to the scripts, but they still give a good impression of how the system works – he differences aren't substantial.

Compability

The converter deals with standard Latex as described in Leslie Lamport's *Latex: User's guide and reference manual* and a small handful of packages, notably the `natbib` package (Patrick Daly, *Natural Sciences Citations and References*) and some graphics packages.

The script also deals with the compilers managed by MiKTeX – LuaLaTex, XeLaTex, and pdfLatex.

As to InDesign, the scripts can be used from InDesign CS6.

Features

The items listed here are not fixed, they will probably change over time as missing features are added.

What is converted

– \newcommand (and \renew- and \providecommand and \def) statements are processed.

– Accented letters, Greek, and many math characters are processed. Not all math symbols are included but it's easy to add any missing characters.

– Many equations are converted. They appear as normal text and can be styled. The original Latex code is stored at display equations and can be retrieved easily in case of problems. Equation arrays are not (yet) converted. Problematic equations can be handled with a separate script (see pp. 23–28).

– During the conversion, tables and figures are placed as anchored floats on the pasteboard and can be placed later.

– Figure scaling (relative and absolute) is supported.

– Figure and table numbering is handled by InDesign's paragraph numbering.

– Cross-references are converted to InDesign cross-references. They update automatically.

– Various list types (itemise, enumerate, description) are processed and numbered with InDesign's paragraph numbering. Newtheorem items are processed as well.

– Instances of `tabbing` are converted to simple tabbed lists.

– Footnotes are converted to InDesign footnotes. Footnote marks are converted to footnotes because there is no InDesign equivalent.

– Margin paragraphs are converted to custom anchored text frames and placed in the outside margin.

– Index entries are converted to InDesign topics and page references. This is largely unproblematic but page ranges need some attention in the form of a script.

– Latex citations (`\citep{...}`, etc.) are converted to numbered citations or text citations if a .bib file is present.

– Hyphenation exceptions are processed. They're applied in the document or added to the exception dictionary.

– Comments are moved to anchored text frames on the pasteboard.

And what is not (yet)

Some (maybe all) of the missing items listed here will be implemented later.

– Not all equations are converted. They are highlighted in the text. Equation arrays are (still) ignored (on purpose), and it's likely that there are certain equations that the script doesn't handle. These can be handled using the Latex2Ind script (see pp. 23–28).

– Only one parameter of figures is handled: sizing (both percentages and absolute values). Other possibilities (rotation, cropping, etc.) are not (yet) supported.

– Over- and underlining isn't handled.

– The picture environment, used for line drawing, isn't supported at all.

– Only one bibliography format is used to generate a document's bibliography (natbib). The text explains how you can add other styles.

InDesign pain points

Two InDesign problems can make things difficult when converting Latex files: math and character styles.

The Math problem was mentioned earlier. It comes in two guises. Most glaringly, of course, InDesign doesn't have an equation editor. InDesign 2024 introduced rudimentary equation support, and Adobe announced a fully fledged editor for the near future. Whatever form that editor takes, the equations will be in a graphic format (SVG).

Text equations – that is, equations that consist of normal text – can be achieved only with movemen's the MathTools plug-ins and with the current converter. MathTools is (still!) superior to what the script can handle, but at the same time the two solutions aren't mutually exclusive. The script will probably never be able to handle everything that MathTools can.

The second pain point is character styles – to be precise, InDesign's inability to handle overlapping character styles. This means that if you use a character style to apply subscript and another style to apply bold, you cannot apply both if you want a subscripted bold characer. Whichever style is applied last, wins (see p. 7 for more discussion).

The upshot is that character styles are needed for individual attributes and for each possible combination. In our example, for instance, we'd need a character style bold-subscript in addition to the bold and the subscript styles.

(You're wondering why we use character styles for bold and superscript? Often a different typeface is needed for bold; as for superscripts, we need subscripts to subscripts, so we can't use InDesign's subscript.)

Tantalizingly, when character styles are applied using GREP styles those character styles *are* accumulative. But heavy use of GREP styles still slows down InDesign considerably and should be used judiciously.

These are the two main problems. Other pain points are caused by less problematic shortcomings (the lack of a useful end-of-range marker for indx page references; see p. 60) and bugs (InDesign's handing of numbered paragraph styles in frames on the same page; see p. 36), but they can be handled by scripts.

Installing the scripts

The downloaded scripts are in a zip file that you should unpack in InDesign's Scripts Panel folder. Do as follows.

Open the Scripts panel (Window > Utilities Scripts) and right-click the User folder (Figure 1). This opens the folder in the operating system's default file manager (Explorer on Windows, Finder on MacOS).

Now extract the files in Latex-to-InDesign.zip into the `Scripts Panel` folder. When you click in InDesign's Script panel it will be updated; see Figure 2.

When installed correctly, the panel shows one folder added to the panel (`Latex-to-InDesign`). Expand that folder to show the scripts in the folder. You'll notice two additional folders, `BST` and `Includes`.

The `Includes` folder contains scripts that are called by the converter (`10-latex-to-indesign.jsx`). You wouldn't normally open that folder, the scripts in there can't be run independently.

The `BST` folder contains scripts that are (aspire to be, really) the equivalent of Latex's `.bst` files. These are bibliography style files which are used to format a generated bibliography. See "Citations and bibliography" for details.

The scripts with numbered prefixes are the ones you'll run – most of them anyway. A short description of each follows.

`010-latex-to-indesign.jsx` This is the main converter. It runs against a Latex text file that was placed in an InDesign document. It converts special character (which in Latex is everything that's not on a UK/US keyboard), figures, tables, math, footnotes, cross-references, index markers, etc. etc.

`020-number-captions.jsx` All captions are numbered in a separate pass. Numbering is handled by InDesign's paragraph numbering, the script sets the numbering in the relevant paragraphs, captions, section headings, etc.).

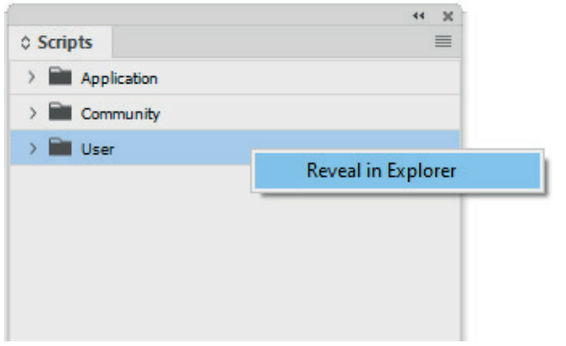

Figure 1. Open the Scripts folder

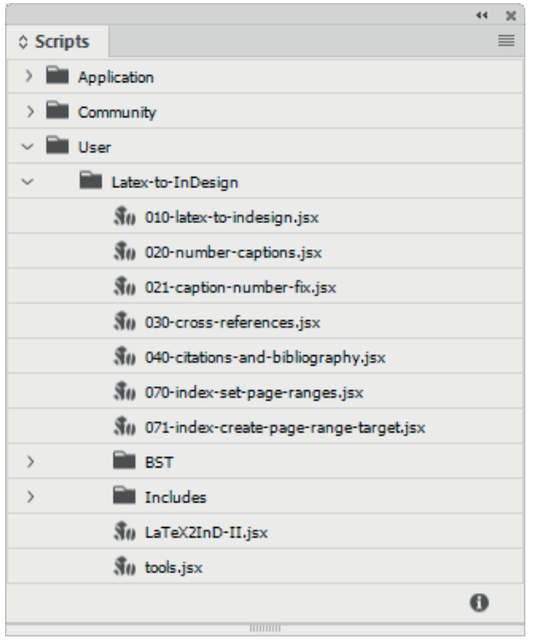

Figure 2. Installed scripts

`021-caption-number-fix.jsx` When two (or more) figures or tables appear on the same page, the numbering is off because of an InDesign bug. This is fixed by the script.

`030-cross-references.jsx` Latex's `\ref` and `\label` commands are placed in anchored frames on the pasteboard, to be processed later.

`040-citations-and-bibliography` Citations in the text are resolved as either bracketed numbers or as full text references. The Latex .bib file must be present (see "Citations and bibliography"); the preamble, too, must be present with the required entries.

`071-index-set-page-ranges.jsx` This script is run only immediately before generating the index. It sets page ranges at the relevant index markers.

`071-index-create-page-range-target.jsx` Latex's page ranges are handled by targeting a bookmark that indicates the end of the range. InDesign can't do that. The script creates a workaround.

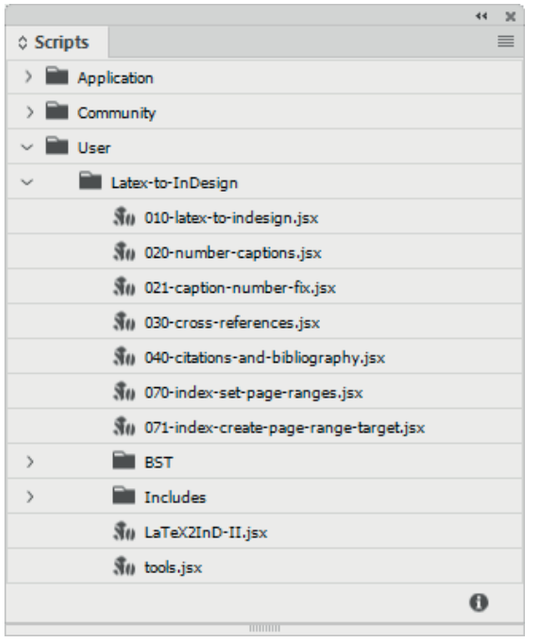

Figure 3. Figure 2. Installed scripts (*repeated*)

Preparation

Before you run the scripts you need to set up a document and organise the environment.

Set up a document

There's great variety in how a document's properties can be specified in the Latex code – paper size, margins, type size, number of text columns, etc. etc. For that reason the script doesn't try to get document specifications: you'll have to create it yourself.

If your document specifications define the text area using the four margins, you can use InDesign's native New Document window. But if the type specifications define the size of the text area and its location on the page (using top and inside margin), it's convenient to use the Page set-up script. (This script is also a good replacement for InDesign's New Document window because it sets the baseline grid as well.)

For the converter there are just two important things. First, create a paragraph style `Body` for the main text. Set at least the font, point size, and leading – and first-line indent, if needed. And if the Latex file contains figures, and if the figure width is specified relative to the text width, then, naturally, the width of the text frame must be set correctly.

If you add hyphenation exceptions to the exception dictionary (see "Hyphenation exceptions" on page 51), and if the document's language is not the same as the application's language, then you must set the language in the paragraph style.

Finally, if you don't use ComputerModern for the text, create a character style `Math` (at the top level, i.e. not in a style group) and set a math font in it. Use a real math font such as Cambria Math, Fira Math, or STIX.

Do not create any other styles: the script creates all the styles on the fly, basing all these styles on the `Body` style – including a `Body (no indent)` style for use after headings, display material, etc. See page 7 for details.

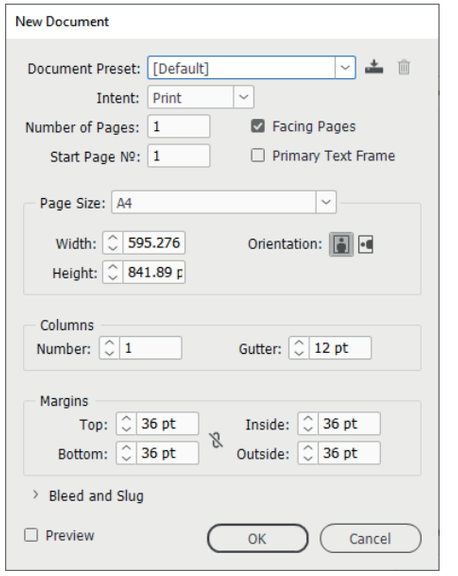

Figure 3. InDesign's page set-up window

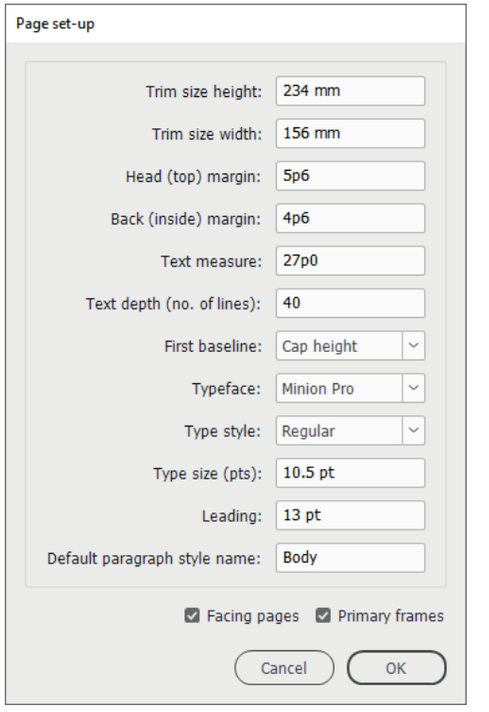

Figure 4. Scripted page set-up

Figures

Figures should be placed in a subdirectory of the InDesign document's directory and this subfolder should be named `Links`. The converter ignores path names in the Latex code so you can leave them there. (This is not entirely convenient and will be changed in a future release.)

Citations and bibliography

If you want Latex-style citations (e.g. `\citep{...}`) to be resolved, you need to place the .bib file in the InDesign document's folder. In addition, the preamble should include a natbib package reference that states how citations should be handled, for example,

```
\usepackage[numbers, sort&compress]{natbib}
```

Finally, if a bibliography should be generated, the preamble should state its format:

```
\bibliographystyle{unsrtnat}
```

See "Citations and bibliography" for more details on citations and the bibliography.

Preamble

The script looks for the `preamble.tex` file. `\newcommand`, `\renewcommand`, `\providecommand`, and `\def` defined in the preable are applied to the text. These four can be in the document as well.

Hyphenation commands in the preamble are added to InDesign's hyphenation exception list. Hyphenation commands in the document are applied only in the document. See "Hyphenation exceptions" for details.

Styles

The script creates all styles on the fly: character, paragraph, object, cell, and table styles are created as needed. All styles are based on the paragraph style `Body`. This is the only style that you should add to the document; the only properties you should add are font, point size, leading, and first-line indent. Setting the Body style correctly is important, especially in the case of equations, which are based on that `Body` style, and equations are difficult to recompose after a type change later on.

In each style panel a style group `~~Lmath` is created, which is used for styles that are used for math only.

The generated style names are named for the commands: the paragraph style used for section headings is `section`; the one for subsections, `subsection`; etc. The created styles have little content, you should specify them afterwards. For instance, in the chapter and the three section styles the chapter and section numbers are defined and some space before and after. You'll need to set the correct font and type size and spacing yourself. Figure 5 shows all the styles that the script creates.

Latex doesn't have character styles (though they can mimicked by `\def` and `\newcommand`). The script creates character styles for all character formatting, using the command names for the style names. Thus, for `\emph` items the script creates and applies a character style `emph`; for text in `\textbf` a character style `textbf` is created.

Latex's 'character styles' are accumulative in the sense that when you apply bold to some text, and italic to part of that bold text, you get bold–italic; so for bold, italic, and bold–italic you need only two styles: bold and italic. You get bold–italic by applying both bold and italic; see Figure 6.

Unfortunately, InDesign doesn't work like that. When you apply bold to some text that's already italic, you end up with bold, not bold–italic, so bold–italic needs a separate character style. For an example like this it's not such a problem. But in other cases it can get quite ridiculous. For example, each of the various combinations of sub- and superscript needs its own character style, as shown in Figure 7. As you can see, that

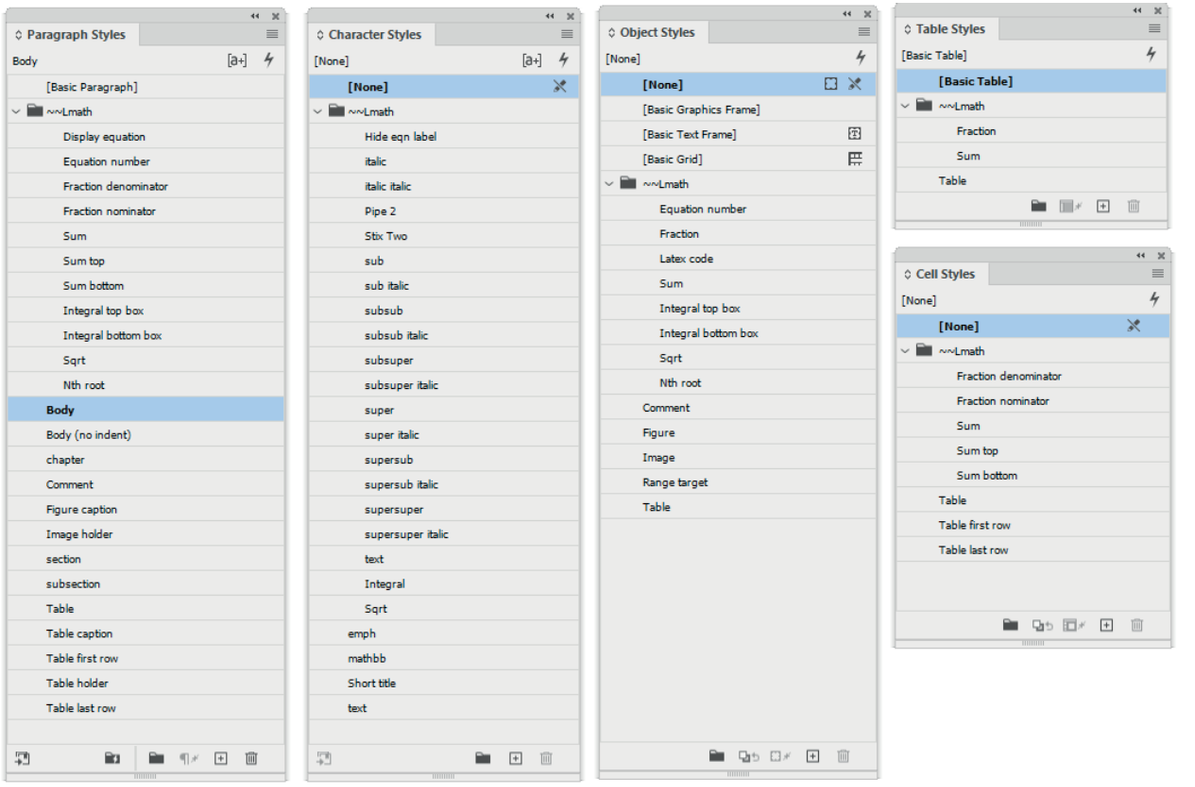

Figure 5. Styles created by the script

```
\textbf{bold}, \textit{italic}, and \textbf{\textit{bold-italic}}
```

bold, *italic*, and ***bold–italic***

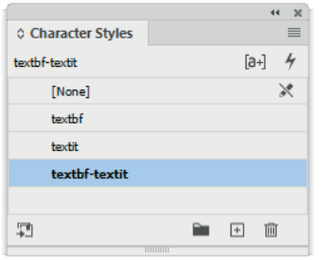

Figure 6. InDesign doesn't have Latex's accumulative styles

can lead to an explosion of character styles, but unfortunately that's the way things are (and will stay; accumulatove character styles is one of the most frequently requests, but Adobe aren't interested).

The character styles for all the super and subscripts, therefore, don't use InDesign's super- and subscript formats, but set them all using vertical movement and type scaling.

So you see that the preferred way to convert a document is to start with a new document. Create a paragraph style `Body` and set its point size and leading. Then run a conversion so that the script creates all styles. Fine-tune the styles and remove all content and you have a template you can use again and again. Re-place the raw Latex code and run the conversion again.

Figure 7. Profusion of character styles

Latex to InDesign

This chapter is a quick tour that runs through all aspects of the conversion so that you get a feel of what is involved. Details of each operation follow in separate sections.

Place the .tex file

Make sure you prepared everything as shown in "Preparation" on page 5. Open the InDesign document you created and place the Latex file. If you used the PageSetup script (see page 10) then pages will be added automatically to flow all the text into the document. Open the Scripts panel.

Note: Make sure that line endings are hard returns (carriage return + line feed). Unix and Mac files (may) use line feeds only (aka as soft returns). After placing the file, show the hidden characters (Type > Show Hidden Characters). Hard returns show as blue pilcrows (¶), soft returns as blue negative signs (¬).

To change soft returns to hard returns, just replace `\n` with `\r` using the GREP tab in the Find/Change window.

Run the converter

Run the `010-latex-to-indesign.jsx` script from the Script panel. This script does most of the conversion. If the tex file contains a lot of figures and equations it may take some time to finish.

Tip: The script doesn't have a progress bar, but you can get a sense of what's going on by opening the GREP tab of the Find/Change window. All the regular expressions will zoom past. And to get an extra sense of what's going on, open all the style panels so that you can see the styles being created in real time.

Inspect the file

When the conversion is done, inspect the file. We'll first go through the sample file and explain what the different colours mean and why the

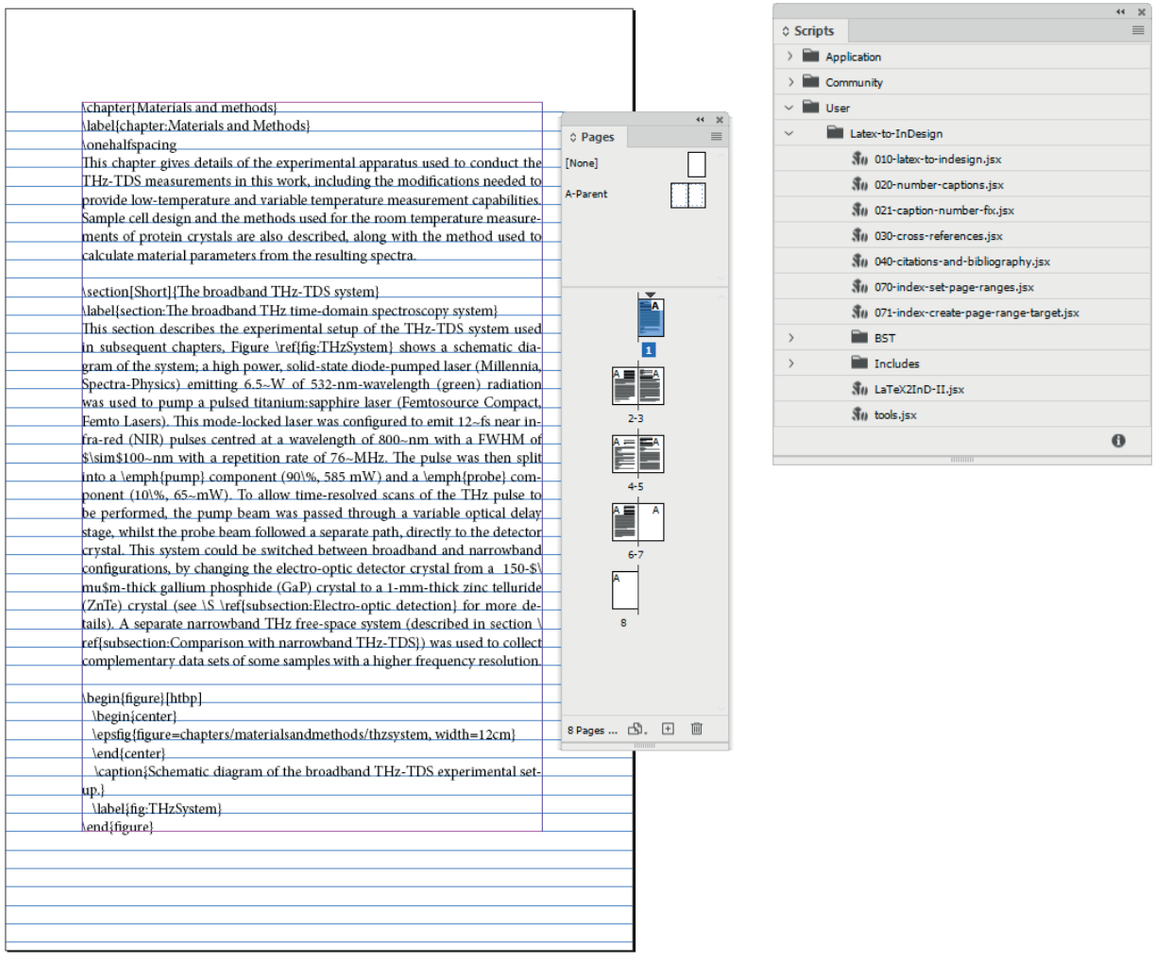

Figure 8. A .tex file placed in InDesign

document is in the state it is (see Figure 9), then we'll go through the scripts needed to finish the document.

Text condition

The script created a text condition – `Math` – for all math in the text (single characters and equations); the highlight colour is yellow. This is just to highlight all math, it has no other function and can be removed from math that turned out well.

Cross-references

The script moved all `\ref` and `\label` commands to anchored frames on the pasteboard (shaded light red). Cross-references are resolved by a separate script because in book files it makes sense to resolve the cross-references after all the book's documents have been converted.

Figures and tables

Figures and tables are placed as anchored text frames so that they stay close to their text references. After the script has finished you'll need to place the figures manually as floats.

Caption numbering is handled by InDesign's paragraph numbering using a dedicated style. The style is applied by a separate script because that proved to be better.

Chapter and heading titles

Like captions, chapter titles and section headings are numbered by InDesign's paragraph numbering using dedicated paragraph styles.

Comments

Comments are placed as anchored text frames on the pasteboard (Figure 10). The % sign is left so that it's clear that it's a comment. The frame is shaded light blue.

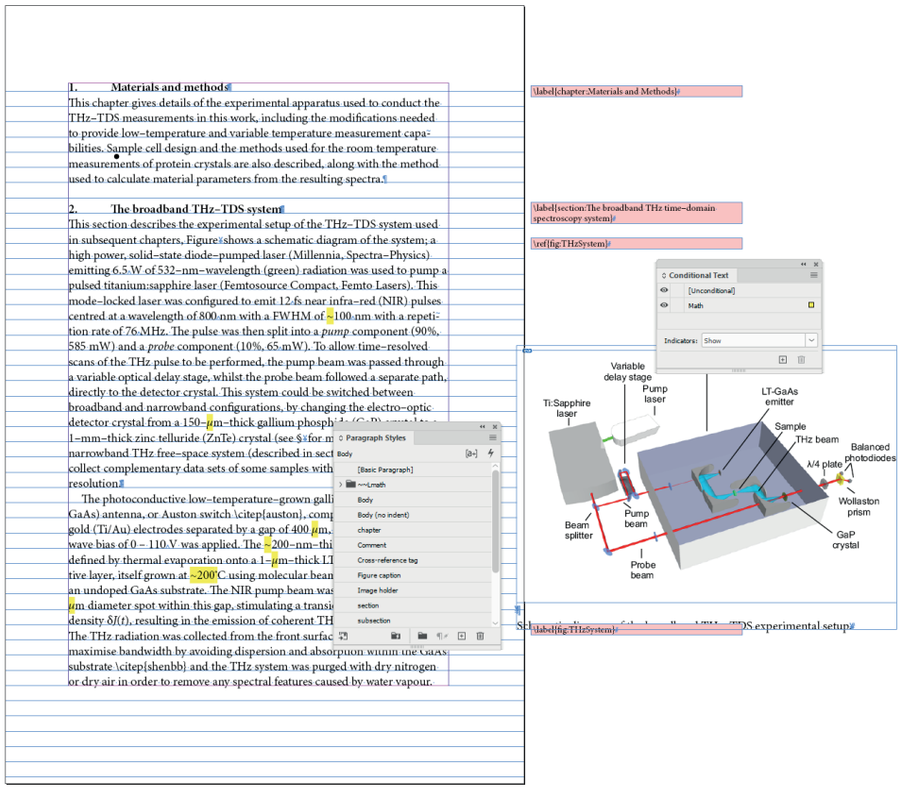

Figure 9. After running the first converter script

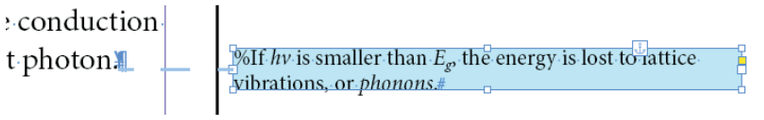

Figure 10. A comment

Clean up the document

The sample document shows a disturbing but innocent bug in InDesign's display (see Figure 11): sometimes a text condition's colour stretches all the way down to the bottom of the screen. The same happens with the pink colour applied by InDesign to missing characters.

Ignore the problem for the moment and check whether the math was converted correctly. If it is, simply delete the text condition. If there are any math problems that you want to keep highlighted, remove the text condition from everything that's correct; to remove a condition from some text, select the text and apply `[Unconditional]`.

The pink bars, too, are a display bug (their height, not their presence). In Figure 11 there are two characters that aren't present in the applied font, which InDesign shows as pink boxed crosses. The problem is that the pink marker colour is streched to the bottom of the screen, but since you'll fix those missing characters anyway the problem will go away.

Missing characters are probably math. The script should have caught most of them by looking for the Math character style (in which you specified a math font), but other characters, too, may be missing from the text font. You can run a separate script to fix any missing glyphs (Figure 12).

Place figures and tables

Figures are text frames that contain an inline graphic and a caption; tables are text frames that contain a table and a caption. They are initially placed as anchored text frames. Their anchors are placed where the Latex code was. They're placed as anchored text frames so that they move along with the text while you place those anchors as floats, which will reflow the text.

When you're ready to place the figures and tables, select the first one and release it (so that it becomes an independent frame). To release a frame, select it, then do Object > Anchored Object > Release.

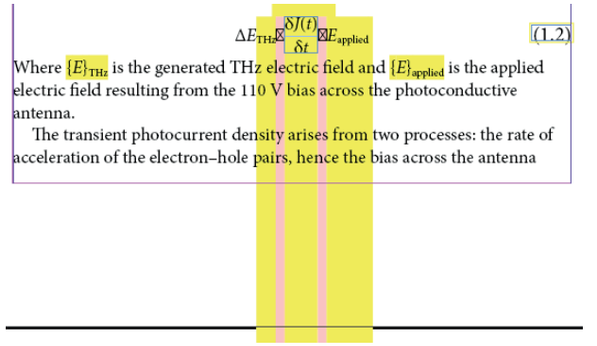

Figure 11. Display mess

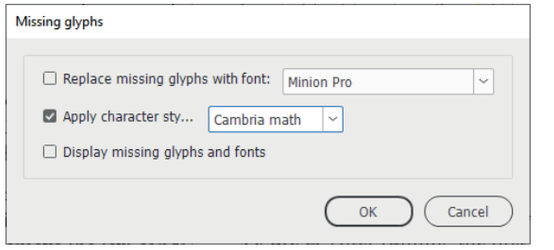

Figure 12. Fixing missing glyphs

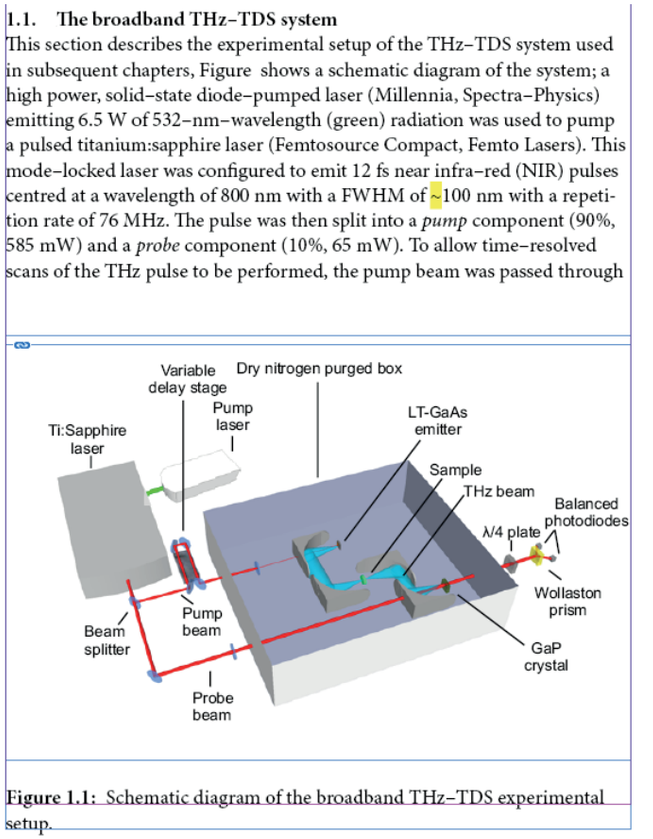

Figure 13. A figure placed as a float, sections and captions numbered

Number captions

To number all captions, run the `020-number-captions.jsx` script. For details on captions see page 35.

If there are two or more figures and/or tables on a page, the caption numbering will be off. This is an InDesign bug. The order of numbered items on a page is determined only partly by their location: if all pages contain one figure or table, the numbers are correct, but on pages with two or more figures or tables, the numbering is wrong. The bug is that InDesign tracks numbered paragraphs in different stories chronologically, not geographically, so to speak.

So for InDesign, the paragraph's location on the page doesn't matter, it's the order of creation. The problem occurs because the converter must work from the end of the document to the start. If the problem occurs in your document, run the `021-caption-number-fix.jsx` script to fix it. See page 36 for details.

In section numbering you won't see this problem because in a document all section numbers are in a single story. In book documents the chapter numbers are handled by the Book panel's number updates, so these aren't a problem either.

Cross-references

Latex's `\ref` and `\label` commands were placed as anchored text frames on the pasteboard. The cross-references aren't resolved yet, that's handled by the separate script `030-cross-references.jsx`. The separate step is needed because you may have a book with several chapters, and cross-references should be resolved when all document are available.

TOC items

The table of contents and lists of figures and tables should be defined manually in table-of-contents styles, and then they're handled correctly by InDesign. Latex's short-title feature can be handled, see page 54 for details.

Index items

Latex index items are converted to InDesign index topics and page references. This is largely unproblematic, but there's one Latex feature that InDesign doesn't support, namely, end-of-range markers. Most word processors have such a feature: some marker to indicate the end of a topic's page range. In InDesign, the end of a range is marked by setting a number of pages or paragraphs after the topic marker, which is very awkward.

The converter emulates Latex's behaviour by placing a mark at Latex's end-of-range marker. Later, immediately before you generate the index, you should run the script `085-index-set-page-ranges.jsx` to apply the ranges in InDesign's index. The script should also be run after any changes that affect the document flow. See page 59 for details.

Citations and the bibliography

This script can replace `\citep{...}` and `\citet{...}` items (and their variants `\citepalt`, `\citetalt`, etc.) with either numbered citations (`[1]`, `[2]`, etc.) or with full text citations (`Smith 2019`). This works only in the presence of a .bib file and if the preamble is set up correctly. See "Citations and bibliography" for details.

Missed and missing items

Not everything Latex has been implemented yet, so the converter will have missed an assortment of Latex commands. And commands imlemented by many packages aren't supported.

To finds remaining Latex look for `\\\w+` in the Find/Change window, using the GREP tab. This matches backslashes followed by letters, numbers, and underscores, in other words, all remaining Latex commands.

Many commands from packages can be converted by scripts, though the extent to which this will work depends on what is actually implemented. These range from very simple things like `\toprule` (introduced by the booktabs package), which is simple, to the commands added by the Ti*k*Z drawing package, which will be hard to handle.

Another thing to look out for is various special characters. It usually pretty easy to write a script to handle various formats for special sorts, such as the commands for phonetic character introduced by the TIPA package. See "Creating additional character-conversion scripts" for how to add these to the converter.

Chapter and section numbers

The script looks for the Latex items `chapter`, `section`, `subsection`, and `subsubsection`. It creates paragraph styles on the fly for these heading types and applies them. Paragraph numbering is set and some formatting is applied, but undoubtedly there is some fine-tuning to be done after the conversion.

Chapter and secion numbering is handled by setting the section levels in the styles. The number strings depend on whether the file contains a chapter heading. If the file needs a chapter number, you need to set it manually (this can be done after the conversion, or you can choose to use automatic numbering). You set the chapter number in the Numbering & Section Options window (right-click a page and select Numbering & Section options from the context menu); see Figure 14.

In the Paragraph Style Options window, in the Bullet and Numbering tab, the relevant parts are `Number` (the number format) and the number's position. In the `Format` dropdown you select the number format as usual. At `Number` you enter the number string (the number string can contain words such as Figure or Table; see p. 35 for details).

The number itself is composed using the special characters `^H` for the chapter number, `^#` for the current number, and `^n` for level placeholders. Other formatting characters can be inserted as usual, as in the table, below, where `^t` stands for the tab character. All these can be accessed by clicking the small black triangle next to the Number dropdown.

	With chapter title	Without chapter title
chapter	`^#.^t`	–
section	`^H.^#.^t`	`^#.^t`
subsection	`^1.^#.^t`	`^H.^2.^#.^t`
subsubsection	`^1.^2.^#.^t`	`^H.^2.^3.^#.^t`

The levels are set as well. Chapter headings are always level 1. If there is a chapter, section headings are level 2, without a chapter heading the section is level 1. Et cetera.

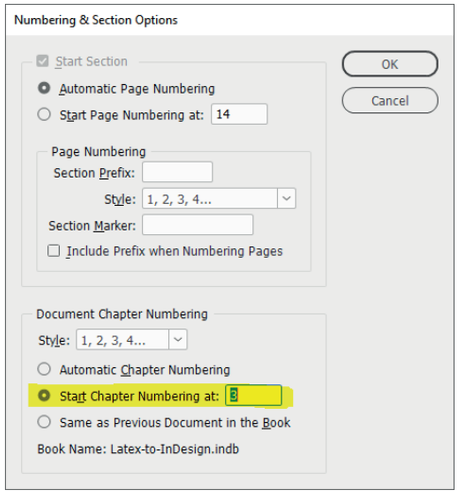

Figure 14. Numbering and section options

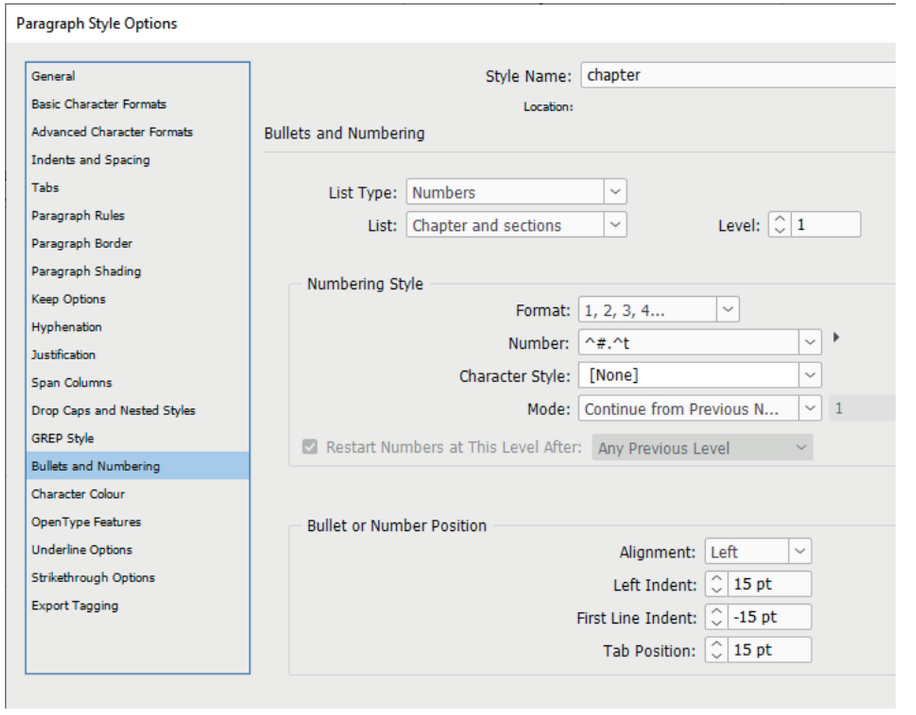

Figure 15. Paragraph numbering

The script also sets the left and first-line indents and the tab position to create hanging indents. All these values should be fine-tuned.

Equations

InDesign doesn't have any native equation processing as text objects, but with a variety of scripts and styles you can get a long way. Figure 16 shows a page with equations that were converted by the script from Latex code to InDesign.

All these equations are normal text. They're constructed using tables (fractions, Sum and Prod constructs in display equations) and/or anchored text frames (sum, integral, square root). So everything in the equation is accessible. Everything is in styles (character, paragraph, cell, table, and object styles) which are created on the fly. They are based on the `Body` paragraph style, so it's important to set that style before you run the script (see "Set up a document" for details). It's possible to change the document's type size and reformat the equations, but that's always a bit of a hassle: better to get it right from the start.

There's one type of equation that's not yet supported, namely multi-line display equations in Latex's `eqnarray` environment.

But apart from those, inevitably there will be equations that the converter can't handle. If the equation doesn't look too complicated you can just try to fix it manually.

If that doesn't work, you can try InDesign's equation tool (available from ID 2025). It produces a graphic that you place in your document. At the moment of writing (March 2025) InDesign's equation tool supports only MathML, but rumours swirl around that Adobe are looking into supporting Latex as well. There are tools on the web to convert Latex code to MathML, that's another possibility.

For complicated equations you have two options. The first is the MathTools plug-in made by movemen, but that's not a free solution. The plug-in is excellent and, like the scripts described here, equations composed by it are normal text objects. Furthermore, the plug-in is scriptable.

The other option is a solution that was designed by Jaime Gómez Ramírez of the Universidad Autónoma de Madrid. This approach uses an InDesign script that displays a window into which you enter the Latex code, then when you run it, the script creates a PDF of the equa-

An eqn in a paragraph $x = \frac{y + z\,/\,2}{y^2 + 1}$ with surrounding text

A paragraph with a fraction with an embedded fraction $\frac{x + y}{1 + \frac{y}{x + 1}}$ in it.

A simple square root $\sqrt{x + y}$ followed by an nth root $\sqrt[n]{2}$.

And a square root applied to a fraction $\sqrt{\frac{ln\ n}{n}}$.

A square root in a fraction: $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$.

And another fraction. This one should really be a display equation to avoid a paragraph with leading adjustment, $l_c(\lambda_{\mathrm{THz}}) = \frac{\lambda_{\mathrm{THz}}}{n_{\mathrm{opt}} - n_{\mathrm{THz}}}$, but some people don't mind that. We'll have to cater for that anyway. Paragraphs with unequal leading generally don't look too good.

And an in-text eqn in its own paragraph. Looks like a display equation: $R(\nu) = \left|\frac{n_{\mathrm{sample}}(\nu) - n_{\mathrm{air}}}{n_{\mathrm{sample}}(\nu) + n_{\mathrm{air}}}\right|^2$

Display equations, the first two unnumbered, the rest, numbered:

$$\sum_{i=1}^{n} x_i = \int_0^1$$

$$\alpha(\nu) = \frac{2}{d}\ln\left\{\frac{A(\nu)}{A_0(\nu)T(\nu)}\right\}$$

$$\alpha(\nu) = \frac{2}{d}\ln\left\{\frac{A(\nu)}{A_0(\nu)}\frac{[n(\nu) + 1]^2}{4n(\nu)}\right\}. \tag{1.1}$$

$$\alpha_{\max} d = 2\ln\left[D(\nu)\frac{4n(\nu)}{(n(\nu) + 1)^2}\right] \tag{1.2}$$

Some items are sensitive to whether they are in a display equation or in an in-text equation. Compare the sum and integral characters here with the ones in the first display equation, above: $\sum_{i=1}^{n} x_i = \int_0^1$

Figure 16. Sample equations produced by the converter

tion in the background by feeding the Latex code into Latex (which you would have installed previously). For details see LaTeX2InD. Some similar approaches exist but Jaime Gómez's is the most recent, and the script is provided as source code.

In order to make this possible – and because it's always useful to have the Latex code available for troubleshooting – the script stores the Latex equation code at the InDesign equation. The code is stored as a script label in a tiny graphic line (zero-width, 6 points tall) before the first character of the equation (see Figure 17). The line is labelled `Latex code` on the Layers panel. In the text you see it as a blue Yen-symbol (¥) – to make that visible, enable Normal screen code and show hidden characters (Type > Show Hidden Characters).

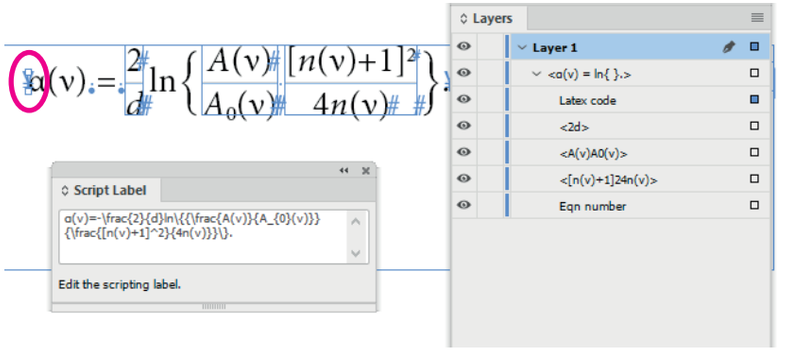

Figure 17. The original Latex equation code is stored in the script label of a graphic line labelled `Latex code`. The line is selected and circled red

Numbered equations

Because of the limitations of InDesign paragraph numbering, equation numbers are in a separate frame, which is anchored after the equation's last character. The frame's anchor is again shown as a blue Yen symbol. The frame is labelled `Eqn number` on the Layers panel. If the equation code includes a label, that label is entered as a script label on the `Eqn number` frame. It's used later to resolve the document's cross-references.

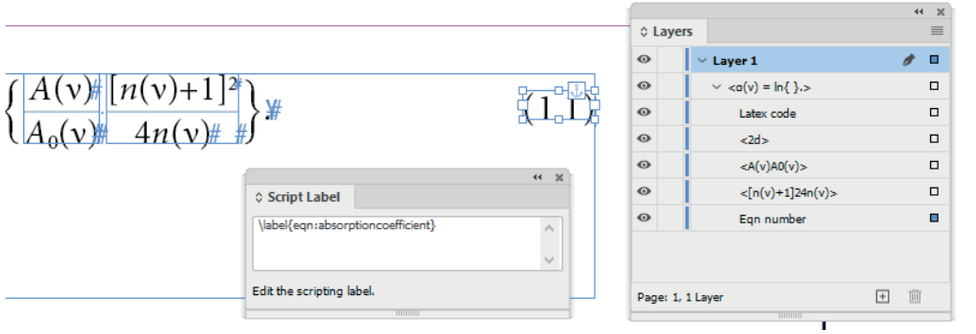

Figure 18. A numbered equation with its (cross-ref) label shown

A closer look at the equation number shows a blue pipe symbol (Figure 19). It represents a zero-width space and is there because a paragraph number shows only if the paragraph has some content. So we must enter something, and a zero-width space makes the most sense.

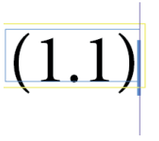

Figure 19. Equation number, with the blue pipe symbol on the right

Some details

As mentioned earlier, equations are constructed using tables and/or anchored text frames. The script creates character, paragraph, cell, and table styles on the fly. To use different parameters in the styles, run a job to let the script create all the styles, then change the styles as necessary, remove all the text, and use this document as a template.

Some details on equation constructs follow.

Inline equations

Some inline equations don't fit the text very well so that it's necessary to change the line's spacing (some would rightly argue that such equations should be done as display equations). The script creates this space, but the result doesn't always look right, as shown in Figure 20, where the fraction's nominator contains a character with an ascender that clashes with a character's descender in the previous line.

The spacing of inline equations can be changed in two ways. To change the space below, select the inline frame and change its bottom wrap (Window > Text Wrap). To change the space above, select the frame's parent character and change the character's baseline shift.

When you change the frame's baseline shift you'll have to adjust the frame's vertical offset so that, in this case, the divider line is realigned with the centre of the x-height (Right-click > Anchored Object > Options).

This change can be scripted. In fact, the converter should check whether clashes occur and adjust the space; this is on the to-do list.

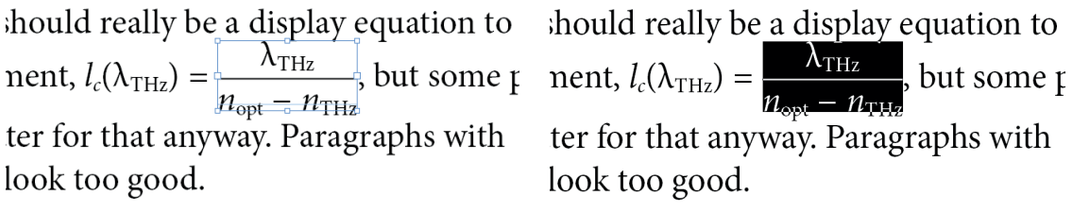

Figure 20. Fix tight spacing of inline equation. Left: the frame selected; right: the frame's parent character selected

Fractions

Fractions are done as tables placed in an anchored text frame. Fractions inside fractions are handled as well (Figure 21). The divider rule is the top rule of the denominator cell set to 0.5 points. The fraction's divider line is aligned with the centre of the line's x-height.

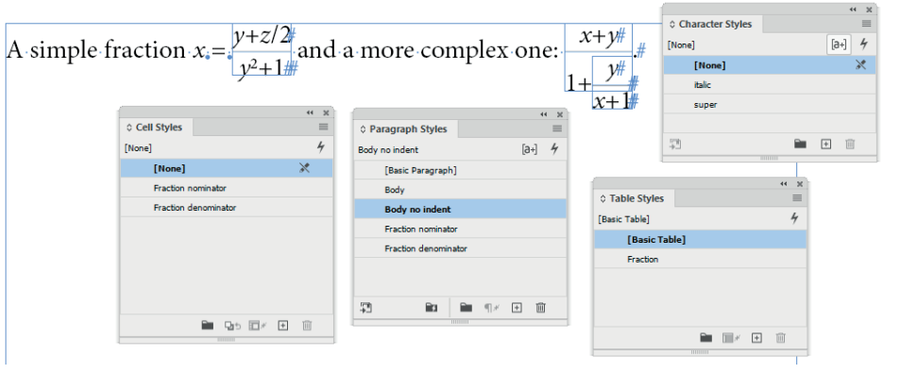

Figure 21. A fraction and the styles created for it

Sum

In display equations the sum symbol and its arguments are constructed as a single-column three-row table. The table is in an inline text frame, its appearance is determined entirely by styles (see Figure 22; again, the panels show only the styles relevant for this construct).

Sum constructs are sensitove to their context. In in-text equations the sum's arguments are rendered as super- and subscripts (Figure 23).

Because the sum symbol is present in most modern OTF fonts, the script uses the character in the current font.

Note: Ignore the blue hash symbols, they indicate the end of a story; cells are stories in and of themselves, that's why you see so many of them.

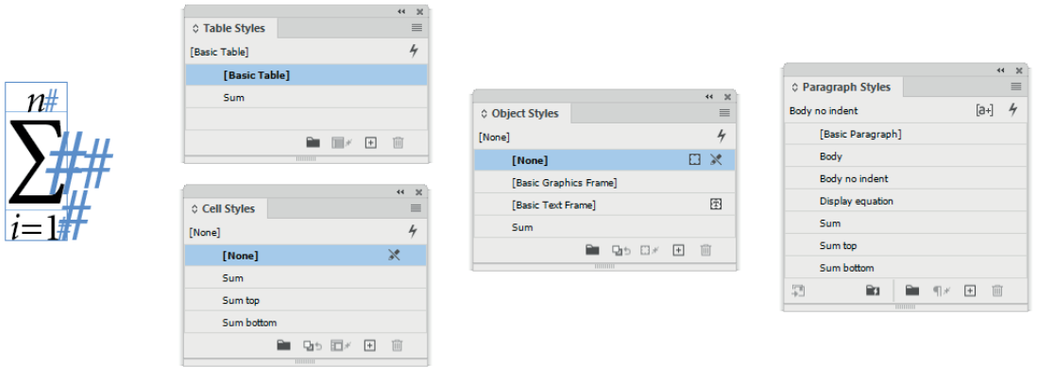

Figure 22. Sum

Product

Product constructs (`\prod{...}{...}`) are handled in the same way as Sums.

Integral

Like the sum symbol, the integral symbol is present in most modern fonts so the script simply uses the current font for it. If the font you use lacks it, change the character style `Integral` and apply a font that does contain it.

Integrals are fiddly. The sample uses Minion as the basic typeface, the character style sets it italic and at 16 points. It also lowers the character by 2 points. You'll probably need to change these values if you use a different font at different type sizes.

The integral's arguments are in anchored frames. Because the arguments need different offsets for precise placements, each has its own object style. The selected top argument, 1, uses `Integral top box`; the 0, the argument at the foot of the integral, uses `Integral bottom box`. These styles, too, probably need to be tweaked when you change the font.

Integrals, too, are sensitive to whether they are in display or inline equations.

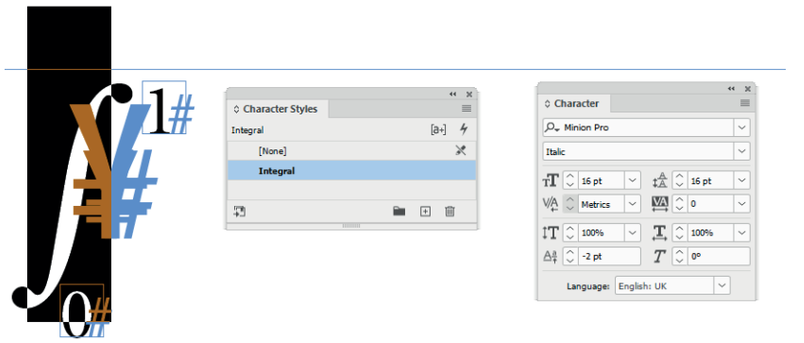

Figure 24. The integral's size is set in a character style

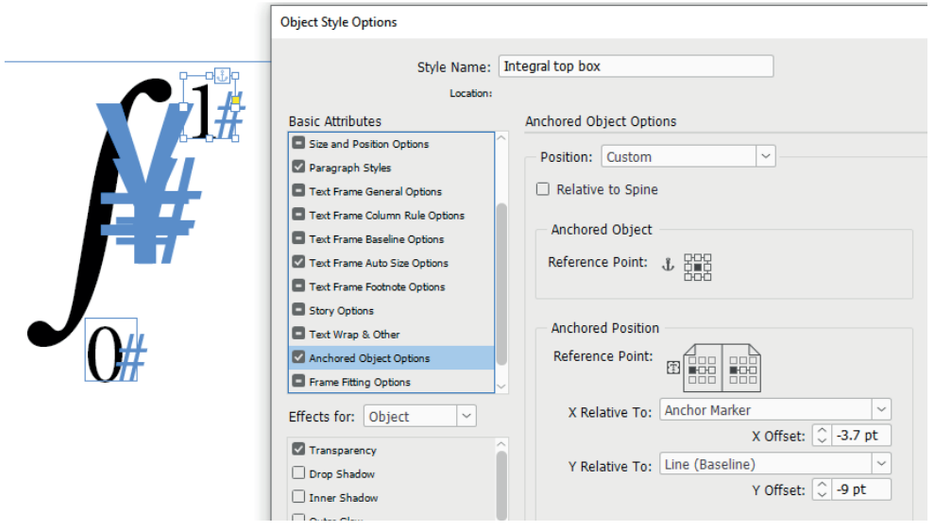

Figure 25. An integral's arguments are in anchored text frames

Root

The argument of a root is in an inline text frame which is labelled `sqrt argument` on the Layers panel. The frame uses a dedicated object style. The root character is in a character style (`Sqrt`) which sets its size and lowers it a bit. This should probably be adjusted to your choice of typeface and point size.

The root character's rule is a paragraph rule set in the `Sqrt` paragraph style used for the argument. The rule's offset and left indent are adjusted to manoeuvre the rule to the correct position.

In other than square roots (n in the example in Figure 26), the n value is in an anchored text frame. The frame's anchor is immediately after the root symbol. The frame is positioned by setting its vertical and horizontal offsets, which should undoubtedly change when you change typeface and/or point size.

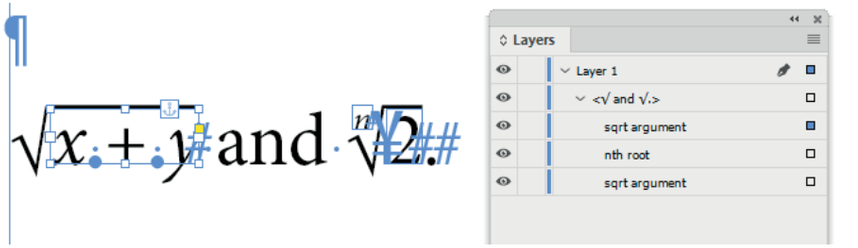

Figure 26. Roots

Limit

The argument of log-like functions is placed as an anchred inline frame (Figure 27).

A inline log-like $\lim_{n \to \infty} x = 0$ with its argument as a subscript and a display equation with its argument below the function name:

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

Figure 27. Log-like functions adapt to their environment

Missing characters

After the conversion you'll probably end up with various pink boxes, which indicate characters that aren't present in the applied font. Often they'll be mathematical characters.

The converter applies the Math character style to missing math characters in the specified Unicode ranges (see Figure 28). In case you see any missing math characters you can run the apply-math-character-style-to-missing-glyphs.jsx script that you'll find in InDesign's Scripts panel. This is in fact the include file from the Includes folder reconfigured to run as a stand-alone script.

The script is configured to target certain Unicode ranges. If you see any missing math or technical glyphs in the text you can change the specified ranges. Maybe uncommenting a range works (remove the two for-

ward slashes at the start of the line), in other cases you might have to set different ranges. You could also change the applied character style.

Equation highlighting

The script highlights all equations using a text condition so that after the conversion it's easy to find them and see whether there are any problems. To remove the highlighting from a single equation, open the Conditional Text panel (Window > Type & Tables > Conditional Text), select the equation, and apply `No Condition`. To remove all highlighting, simply delete the condition.

```
var ranges = [
  '\\x{2100}-\\x{214F}', // Letterlike symbols
  '\\x{2190}-\\x{21FF}', // Arrows
  '\\x{27F0}-\\x{27FF}', // Arrows supplemental A
  '\\x{2900}-\\x{297F}', // Arrows supplemental B
  '\\x{1F800}-\\x{1F8FF}', // Arrows supplemental C
  '\\x{2B00}-\\x{2BFF}', // Misc. symbols and arrows
  '\\x{1D400}-\\x{1D7FF}', // Mathematical alphanumeric symbols
  //'\\x{1EE00}-\\x{1EEFF}', // Arabic mathematical alphabetic symbols
  '\\x{2200}-\\x{22FF}', // Mathematical operators
  '\\x{2A00}-\\x{2AFF}', // Mathematical operators supplement
  '\\x{27C0}-\\x{27EF}', // Misc. mathematical symbols A
  '\\x{2980}-\\x{29FF}', // Misc. mathematical symbols B
  '\\x{2300}-\\x{23FF}', // Misc. technical
  '\\x{1D400}-\\x{1D7FF}', // Mathematical Alphanumeric Symbols
  //'\\x{25A0}-\\x{25FF}', // Geometric shapes
  '\\x{2100}-\\x{214F}', // Letterlike symbols
  //'\\x{2500}-\\x{257F}', // Box drawing
  //'\\x{2580}-\\x{259F}', // Block elements
  //'\\x{1F780}-\\x{1F7FF}', // Geometric shapes extended
];

//More code
```

Figure 28. Specify math/technical Unicode ranges

Latex2InD

As mentioned earlier the converter does not handle certain equations. Some equations are (still) simply too complex for the script and some equations use packages that I don't know about and therefore cannot handle by definition.

These equations can be handled using the Latex2Ind script. The idea behind it is that you feed it the original Latex code, which the script then runs in Latex behind the screens and saves as a PDF file. The script then places that PDF file in the document. This sounds complicated, but once the system is set up it is very easy to use. We'll discuss setting up and using the script in this section.

Installing a Latex compiler

Don't be daunted: this is really quite straightforward. There are various Latex compilers. The system recommended by the author of Latex2Ind (Jaime Gómez Ramírez) is MikTex. You can get it from MikTex's download page. I've used it extensively and it works quite well.

Installation is simple: double-click the downloaded file (basic-miktex-24.1-x64.exe) and follow the instructions. That's all. It's quite possible that you'll never see Latex, it all happens in the background. Nevertheless, in the appendix I give some details because you may need Latex for some troubleshooting.

Installing the Latex2InD script

The script is available at the web site of its author. I adapted it so that it works more conveniently for our purposes – details follow. The adapted script is part of the package that you downloaded from the CreativePro website and you'll find it in InDesign's Script panel (`Latex2InD-II.jsx`), but it's still a good idea to visit the author's web site for more background information on the script and some nice examples of publications for which the script was used.

(*Note*: The script is Windows-only. Only when I was finalising this chapter did I discover that there is a Mac version too. If you're using a Mac you'll have to use the version from the author's web site.)

But before going into the script it's necessary to turn our attention to some details of how Latex uses fonts in equations.

Math fonts

The converter (latex-to-indesign.jsx) renders equations as pure text so that the fonts used in equations match those used in the text. Latex, however, uses ComputerModern in equations if you don't tell it to use something else. In the original Latex2InD script you can't specify any fonts so you'll always end up with ComputerModern. That was one of the main reasons to adapt the script.

Latex2InD now handles fonts as follows. Your document almost certainly contains a character style Math because you used that to fix missing characters in the text, in which you set a math font such as Cambria Math, Fira Math, or STIX. Latex2InD uses that font to create the equation.

That still means that the math font is used for everything in the equation. To have Latex use the document's font for letters and digits in equations, tick `Use the document's font` in the script's window.

Using the script

Select the insertion point where the equation should be inserted and run the script from InDesign's Scripts panel. It shows the interface shown in Figure 29. That's the other main difference with the original script: the original placed the created equation as a float, my adaptation places it as an inline. The script gets the font to be used and its point size from the insertion point.

You enter the Latex code in the panel on the left. You can copy the code there or write it from scratch. It must be well-formed Latex code; enter the code for just one equation.

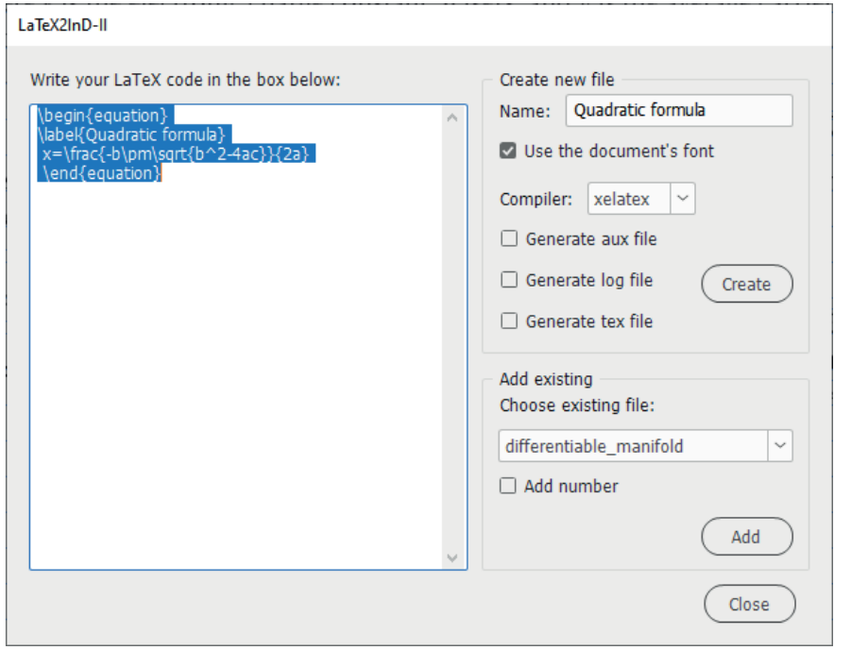

Figure 29. The script's interface

At Name you privide a name for the generated PDF file. If the Latex code contains a label, the label's content is placed in the Name field. You can change the name or leave it blank, in which case the script uses 'equation'. If you use a name that already exists, the script creates a unique name by adding the current date and time to the name.

To use the document's font for letters and digits, check Use the document's fonts.

At Compiler you select which compiler should be used. The script defaults to XeLaTex, and I recommend that you keep this option because this compiler has better Unicode support than the other two available compilers.

To troubleshoot any problems you can tick any of the three Generate checkboxes. You'll find the Latex code that the script generated for Latex to use, and if you know Latex, that code will tell you what the problem is. See page 28 for details.

The generated PDF is stored in a folder equations, which is created by the script in the document's folder. The aux, log, and tex files are placed there as well.

The placed equation

In-text and display equations are in effect the same for InDesign: both are placed at an insertion point and need some manipulation to align them relative to the insertion point's baseline.

The script applies a few settings:

- Skip by Leading is disabled (it's in the Preferences window: Edit > Preferences, the Composition tab). If it's enabled there will be too much space between the bottom of the equation and the following line.
- If the paragraph's leading is fixed, the leading of the character that holds the equation is set to Auto. If it's kept at fixed, the top of the equation runs into the previous line. Auto leading is applied to the character as a local override.
- The script creates an object style (PDF Equation) on the fly and applies it to the equation. The style defines the anchored frame's type (inline)

and enables text wrap, though no offsets are set. If any offsets are needed they are applied manually, which we'll discuss below.

- Latex's `equation` type, which produces a numbered equation, is changed to `displaymath`, which is not numbered. We want to use InDesign's paragraph numbering so that we can use InDesign's cross-referencing.
- Latex's `eqnarray` type, too, is numbered, but now we can't remove the number because the type must be passed on to the Latex compiler. This means that you'll have to crop the equation to hide the number.

Aligning the placed equation

Both in-text and display equations are placed as inlines. All display equations are generated without a number, because we want InDesign to handle the number to ensure correct numbering and in order to be able to handle cross-referencing to the number.

In-text equations

PDF equations, like all inlines, are placed with the bottom of the frame at the insertion point's baseline (Figure 30). Unfortunately, it's not possible for a script to see where the equation's baseline is, that is to say, the vertical position of the *x*, so we'll have to adjust manually the position of the inline frame and its spacing. Three adjustments are necessary, and each is set in a different panel.

First, select the equation's parent character and open the Character panel (Type > Character) to apply some negative baseline shift to move the equation down to increase the space between the equation and the line before it (Figure 31, top).

Second, select the frame and open the Anchored Object Options window (Object > Anchored Object > Options). Apply a negative Y offset to move the equation's line (but not the equation itself) up until the *x* sits on the line's baseline (Figure 31, centre). You can move it until it looks good, or take the exact distance between the bottom of the equation's frame and the *x*'s baseline and enter that in the Y offset's field.

Third, with the equation's frame still selected, open the Text Wrap panel (Window > Text Wrap) and enter a negative value for the wrap's bottom offset until it looks good (Figure 31, bottom).

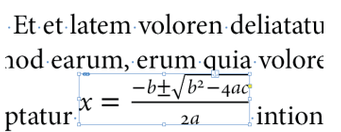

Figure 30. A placed PDF

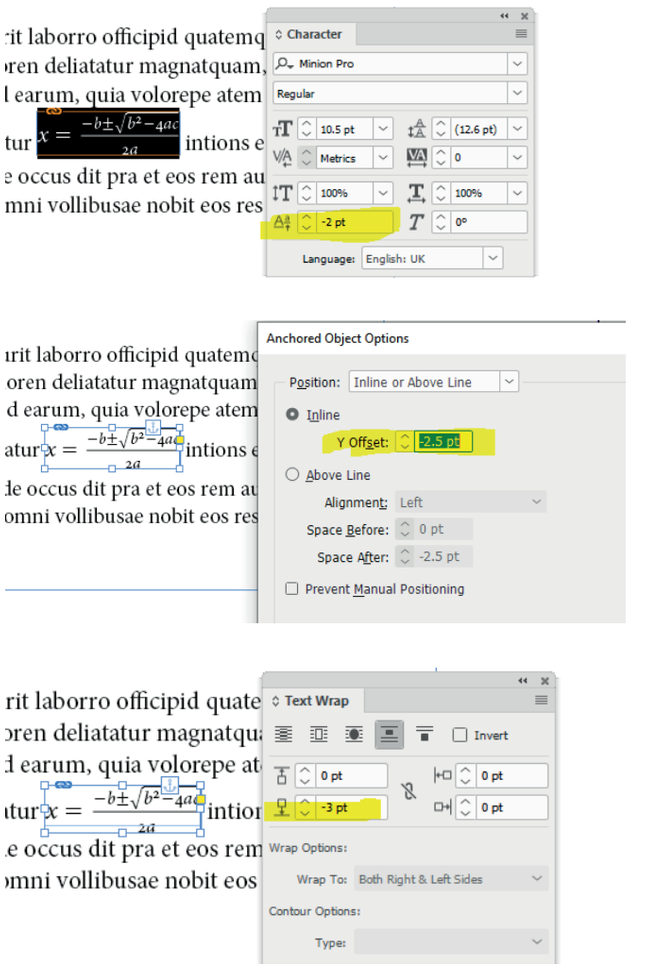

Figure 31. Align and space an in-text PDF equation

Display equations

Display equations sit in a paragraph on their own. Equations of type `equation` and `eqnarray` are numbered using the paragraph style `Display equation` that was created when the document was created. If the equation contains a label, that label is handled like the labels processed by the converter, in a separate frame (see p. 18 for details).

There is in fact very little to adjust, as you can see in Figure 32: all that's needed is to move the number up. As mentioned above, unfortunately a script can't see where the equation's baseline is (in this case, the baseline of *x*) so the number must be aligned manually, but that's very simple.

The number sits in an anchored frame, so you can simply select the frame and drag it up so that the bottom of the frame is aligned with the bottom of the *x*. For precise placement you can place a horizontal guide at the baseline of the *x* and align the number's frame bottom with the guide.

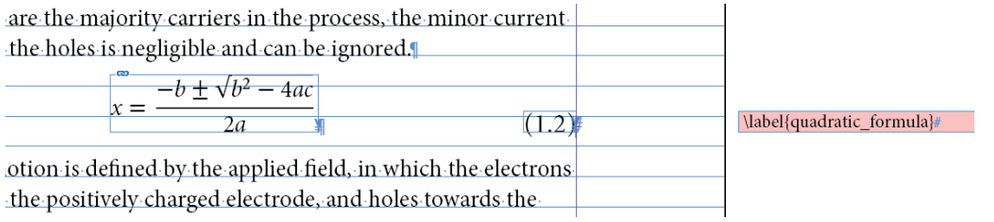

Figure 32. A numbered and labelled display equation

What if the required styles aren't present?

It's not likely to happen, but just in case the required styles aren't present, it's easy to add them. Create a new document, add the `Body` paragraph style and the `Math` character style, then insert a simple equation of type `equation` and make sure there's a `\label{}` in the code so that a number and the label are placed.

Then run the converter, which creates the equation and the required styles. Now select the equation's full paragraph and go to the document to which you want to copy the styles. Make sure there's no selection there and paste the clipboard content. Then press Del immediately to delete the added frame. The styles remain in the document, and now you can use Latex2Ind to create and place a PDF equation.

Placing an existing equation

To place an existing equation, select an insertion point and select the equation from the `Choose existing file` dropdown. There's no information about the type of equation, so if you want to add a number, tick `Add number`. The `Name` field is used for a label.

Troubleshooting

The log file shows any errors encountered, while the generated tex file shows how the script created the Latex code, so you can inspect it. It's then easier to try the code in MikTex's code editor and runner (see 00 for details).

Special characters

Of the hundreds – if not thousands – of characters used in scholarly texts, only the ones that you see on your keyboard can be entered directly into a document. So the term 'special character' captures anything that isn't under a key.

There are six script (include) files that deal with special characters. There is no assumption that these are complete: especially the math file, which contains a list of math characters with their Unicode values, will probably need additions. And the miscellaneous file, too, you're likely to add items to. We'll show how you can add items to these files.

If characters are not present in the used font, InDesign displays them as pink boxes. You can use this script to apply a character style to missing characters.

These include scripts can be run separately as well. They have no dependencies. All you need to do is uncomment the function call. For example, to run the `symbols-math.jsx` script independently, open the file in a text editor and uncomment the first line, that is, remove the two slashes from the first line. All six character-mapping scripts can be run like this. Their names start with `symbols-`.

```
// replaceMathCharacters ();

function replaceMathCharacters () {
```

Figure 33. Comment in the function call

Accented characters

Accented letters are handled by two include files. `symbols-accent-1.jsx` handles accents of the form `\^{o}` and `{\^o}` and `\^o`, where `^` stands for the circumflex accent and `o` for the letter – so they all produce ô. If the requested letter is not present in the Latin-1 Supplement or Latin Extended-A Unicode tables, the script inserts the letter followed by the floating accent. When you use a lot of these floating accents it's therefore recommended that you use a font that contains all those accents, such as the Skolar font family.

The second include file for accents is `symbols-accent-2.jsx`. This one deals with accents in the form `\circumflex{o}`. It too inserts the character followed by the floating accent if the requested character is not in Latin-1 Supplement or Latin Extended-A Unicode tables.

Greek characters

Greek characters are processed by `symbols-greek.jsx`. The character style `emph` is applied to Greek letters. The script handles isolated Greek characters and those that have an argument (`\delta{X}`).

Math symbols

Mathematical symbols are processed by `symbols-math.jsx`. The script contains the code for about three dozen characters, so in a math-heavy document it's likely that you'll have to add characters. This is simple.

Open the script file from the Includes folder in the Scripts panel. It's a text file, so use a plain-text editor. Go to the equivalence table, it starts at line 19. It doesn't matter where in the list you add new items. Do not type the backslash at the Latex name of the symbol.

After the colon add the character to be used in quotes. You can copy and paste a character or you can use the character's Unicode value, which should be entered in the format you see in the list: `'\u0000'`. For example, the approximation symbol can be given as `'≈'` (as in the example) or as `'\u2248'`. The small squares with a question mark indicate that a character can't be displayed by the editor's font. You can change the editor's font or use Unicode values.

The script handles math symbols with parameters, such as `\pm{12}` and `\propto{\frac{a}{b}}`.

When you close the file, make sure that you save it as a text file.

```
19    var pairs = {
20      approx: '≈',
21      ast: '*',
22      bot: '⊥',
23      cap: '∩',
24      cdot: '·',
25      cdots: '⋯',
26      circ: '°',
27      cup: '∪',
28      dashv: '\u22A3',
29      div: '÷',
30      downarrow: '↓',
31      emptyset: '∅',
32      exists: '∃',
```

Figure 34. Fragment of symbols.math.jsx

Miscellaneous symbols

`symbols-misc.jsx` handles a variety of miscellaneous symbols – digraphs, typographic symbols, etc.

The script consists of two parts. The first part is short: it translates Latex's notation for hexadecimal characters to InDesign characters. Latex's format for the hexadecimal value U+014B is `\symbol{"014B}`.

The second part is a long list of Latex code representations and characters that are fed into a simple find–replace function. The code is simple, it speaks for itself. It's therefore easy to add replacement items.

For example, in one of the samples I worked with I found the definitions `\og` and `\og{}` and `\fg` and `\fg{}` (open and closing guillemot, each in two variants, with and without braces). I added them at the bottom of the existing list as shown in Figure 35.

As you can see, some of these replacements are very simple (e.g. replace `---` with en em-dash) while others look more complicated, using regular expressions. When you enter a regular expression, remember that backslashes must be escaped, so that when you enter the expression as a quoted string you should escape the escape characters, which can lead to lots of backslashes.

An alternative to a quoted string is shown in the code fragment on the right. `/\\og(?:\{\})/.source` is the same as `'\\\\og(?:\\{\\})'`, but is easier to read. Especially in longer recular expressions the `/.../.source` format is easier.

Creating additional character-conversion scripts

You can create additional conversion scripts. In the `Include` folder you'll find a script `symbols-ipa.jsx`, which can be used map the TIPA package's strings to InDesign characters. (It could be coded more efficiently, but but at the loss of readability.)

You can use this script as a template for other character mappers.

Like other scripts in the include folder, if you comment in the first line, the function call, you can run it as a stand-alone script.

```
replace ('\\\\l(?![a-z])', 'ł');
replace ('\\\\L(?![a-z])', 'Ł');
replace ('\\\\o(?![a-z])', 'ø');
replace ('\\\\O(?![a-z])', 'Ø');
replace ('\\\\pounds', '£');
replace ('\\\\copyright', '©');
replace ('---', '—');   // em-rule
replace ('--', '–');    // en-rule
replace (/(?<!\\)~/.source, '~S' );   // tilde not preceded by \
replace (/\\og(?:\{\})?/.source, '«');
replace (/\\fg(?:\{\})?/.source, '»');
```

Figure 35. Code fragment of misc-symbols.jsx

```
// replaceIPA()

function replaceIPA () {

   var indd = app.documents.item(0);

   app.findGrepPreferences
      = app.changeGrepPreferences
      = app.findChangeGrepOptions = null;

   var pairs = [
      ['\\\\schwa', 'ə'],
      ['\\\\textinvscr', 'ʁ'],
      ['\\\\textturna', 'ɐ'],
      ['\\\\textglotstop', 'ʔ'],
      ['\\\\textrtaild', 'ɖ'],
      ['\\\\textscg', 'ɢ'],
      ['\\\\texthtb', 'ɓ'],
      ['\\\\textctc', 'ɕ'],
      ['\\\\textcrh', 'ħ'],
      ['\\\\textOlyoghlig', 'ƹ'],
      ['\\\\textbeta', 'β'],
   ]

   for (var i = 0; i < pairs.length; i++) {
      app.findGrepPreferences.findWhat = pairs[i][0];
      app.changeGrepPreferences.changeTo = pairs[i][1];
      indd.changeGrep();
   }
}
```

Figure 36. A custom character mapper

Boxes

mbox

There are three types of `\mbox`: to prevent its argument from breaking across the line end; to allow certain formatting in math environments; and to insert an 'invisible' placeholder character in `eqnarray`s, for instance.

The third one is easy enough. It's an `mbox` command with an empty argument (`\mbox{}`), and since the invisible characters play no role in InDesign, the script simply deletes all occurrences of `\mbox{}`.

The first two are handled together: The script applies `noBreak` to the argument of `\mbox`. This means that in math environments `noBreak` is applied, which strictly speaking is not necessary, but it doesn't hurt either, and treating these two cases the same is more efficient.

No additional styles are needed for `mbox`es; `noBreak` is applied as a local override.

makebox

The argument of the `\makebox` command is placed in an inline text frame without any stroke. An object style is created (`makebox`) which uses the `Body (no indent)` paragraph style. The command's arguments – width and, optionally, alignment – are processed as expected.

framebox

Essentially `\framebox` command is the same as the `\makebox` command with a stroke added to the text frame. From InDesign's perspective, however, things are a bit more complicated, which is why it's handled by a different script.

As for makeboxes, the script creates an object style (`framebox`) for the text frame. The style handles the sizing and appearance of the box. But for reasons I have yet to discover, the object style isn't applied correct-

```
Some \makebox[1in]{boxed} text.
Some \makebox[2.54cm][l]{boxed} text.
Some \makebox[25.4mm][r]{boxed} text.
```

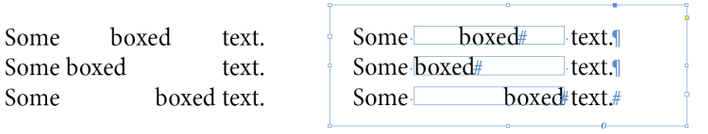

Figure 37. makebox (without and with codes shown)

```
Text in a \framebox[3cm][l]{framebox}
and some in an \fbox{fbox}.
```

Text in a framebox and some in an fbox.

Figure 38. framebox and fbox

ly so that the script has to apply some attributes individually. The script code looks awkward but that's the way it is.

fbox

The `\fbox` command is the `\framebox` command without arguments. The same object style is used as for `frameboxes`, the script sets the text frame's autosize to height and width (the style is created with autosize height only). See Figure 38.

parbox

Paragraph boxes (`\parbox`) are placed as inline frames. The script creates an object style for them (`parbox`) and a paragraph style (`parbox`), based on the `Body (no indent)` paragraph style.

Figure 39 shows the three vertical placement options. I changed the centre placement a bit: if the box contains an odd number of lines, the middle line of the box is aligned with the anchor's baseline; with an even number of lines the centre of the box is aligned with the centre of the anchor's x-height.

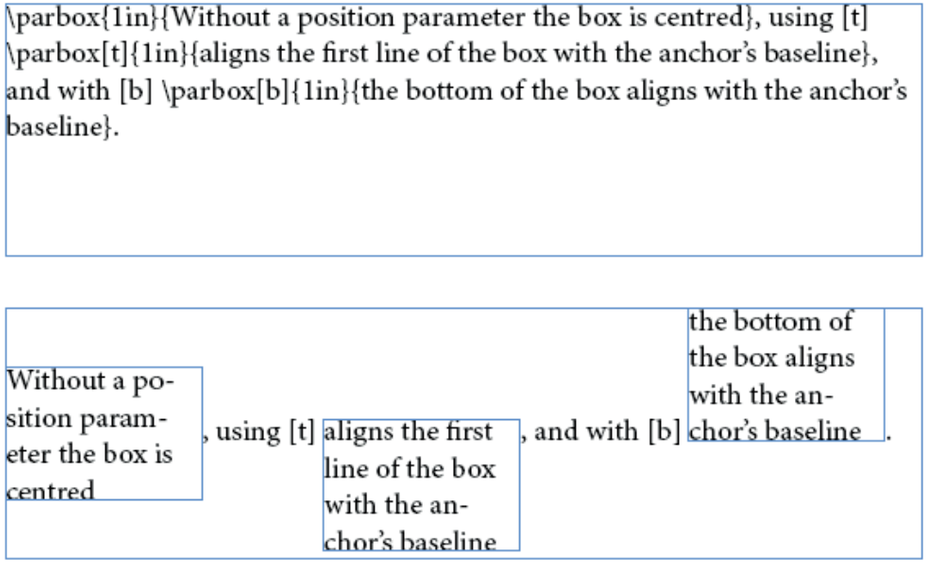

Figure 39. Paragraph boxes

Figures

In InDesign, figures are placed as floats, as in Latex. Initially, however, the converter places them on the pasteboard as custom anchored frames so that when you move them into the text as floats, the unplaced anchored frames travel with their reference. So while you place figures, figures further down in the text remain opposite their preferred location, which makes placing them much easier.

The anchors are placed at the location of the Latex figure code. In Figure 40 the dashed blue line from the figure into the text shows where the Latex figure code was. (To make those lines visible, enable Normal screen mode and select View > Extras > Show Text Threads.) The anchor itself is shown as a blue Yen symbol (¥; see Figure 41; it's partly hidden by the paragraph mark).

Images are expected in a folder named `/Links`, which should be a subfolder of the document's folder. If a graphic can't be found its full path name is placed in the figure rather than the graphic. The script ignores path names in the Latex code to improve portability.

Of a figure's formatting instructions, only scaling is applied, both absolute (as a measurement or as the text width) and relative to the text width. Other instructions such as clipping and rotation are not (yet) supported.

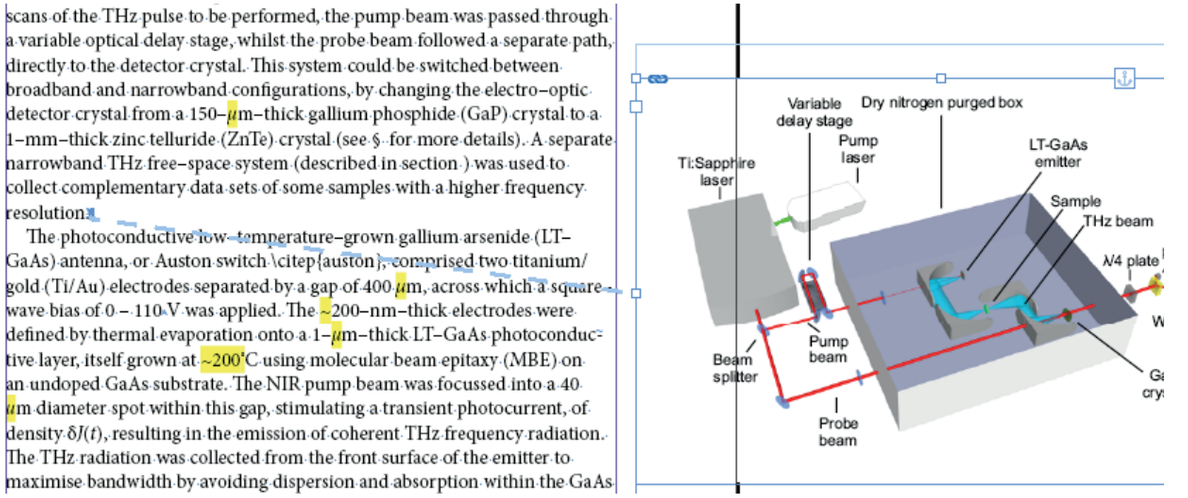

Figure 40. A figure as an anchored frame on the pasteboard

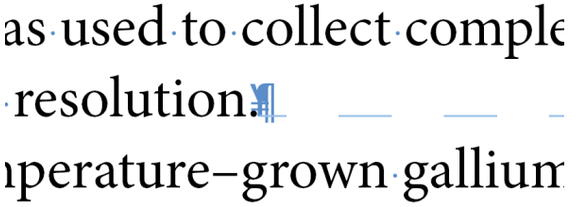

Figure 41. An anchor in the text

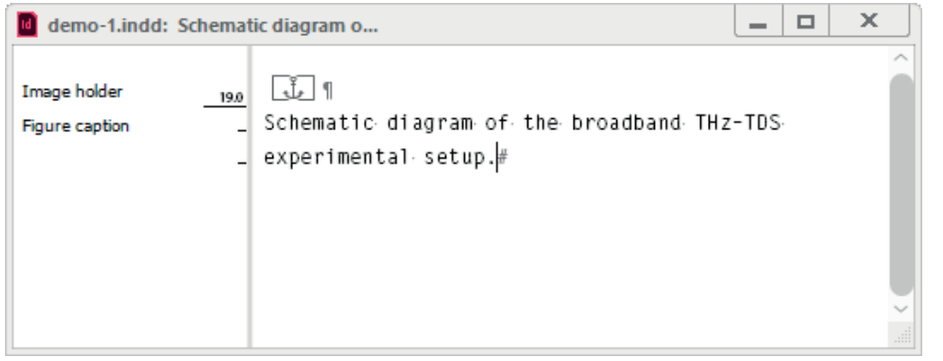

Figure 42. The content of a figure in the Story Editor

Figure composition

A figure is constructed as a text frame that contains a graphic and a caption. The script creates object styles for the container text frame and the graphic (if it's present). The container's object style name is `Figure`, it sets just text wrap at 18 points, which you can tweak as required.

For the image itself the script creates an object style `Image`, which doesn't set anything – just because it's useful always to have a style applied to something so that formatting can be applied later if necessary but also so that you can find them easier later on.

Finally, the image sits in a paragraph for which the script creates the paragraph style `Image holder` (Figure 42). This style has two settings:

point size 0.1 and leading 0. Without these values the gap between the caption and the image would be too big. That distance can be further tweaked by adjusting the caption's leading.

Placing figures

When the text is finalised you can go through it and place the figures as floats. To place a figure, release it so that it becomes an independent frame, then place it at its desired position. (If figures aren't released their text wrap won't work if they are placed before their anchor.) To release an anchored frame, right-click it, select Anchored Object in the context menu, and select Release (or go through the Object menu: Object > Anchored Object > Release). If the document contains many figures it's useful to set a keyboard shortcut to make life easier. In Figure 43 you can see that I applied Ctrl+Alt+Shift+R to the Release command.

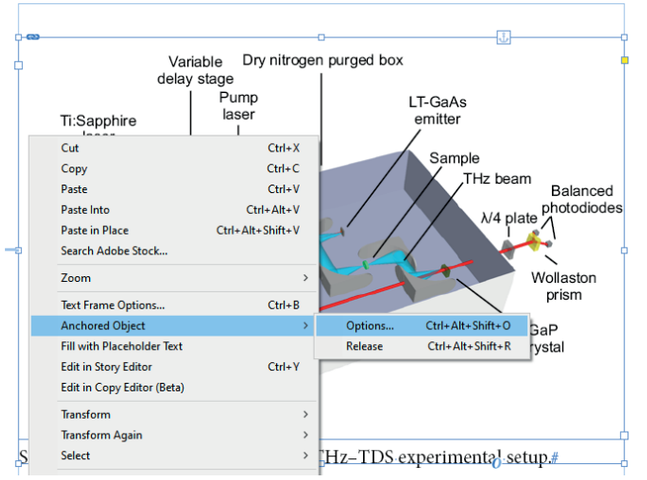

Figure 43. Releasing an anchored object

Column-spanning figures

The column-spanning figures I have seen in Latex files were two minipage environments in a figure environment. The converter places them as two separate figures and groups them.

Captions

The script creates a paragraph style for the captions on the fly (`Figure caption`). Unlike in Latex, InDesign tables and figures need their own paragraph styles because they need their own numbering lists.

Captions aren't numbered during the conversion. Instead, after the conversion run the 020-number-captions.jsx script. It sets InDesign's paragraph numbering on the caption style.

If the document contains a chapter heading, its number is used in the Number string, as shown in Figure 44: `^H` stands for the chapter number, `^#` for the sequential number; the numbering level is set to 2. As you can see, all punctuation is part of the caption's number format (`^>` stands for an en-space). It must be because unfortunately numbering is a paragraph attribute, not a character attribute.

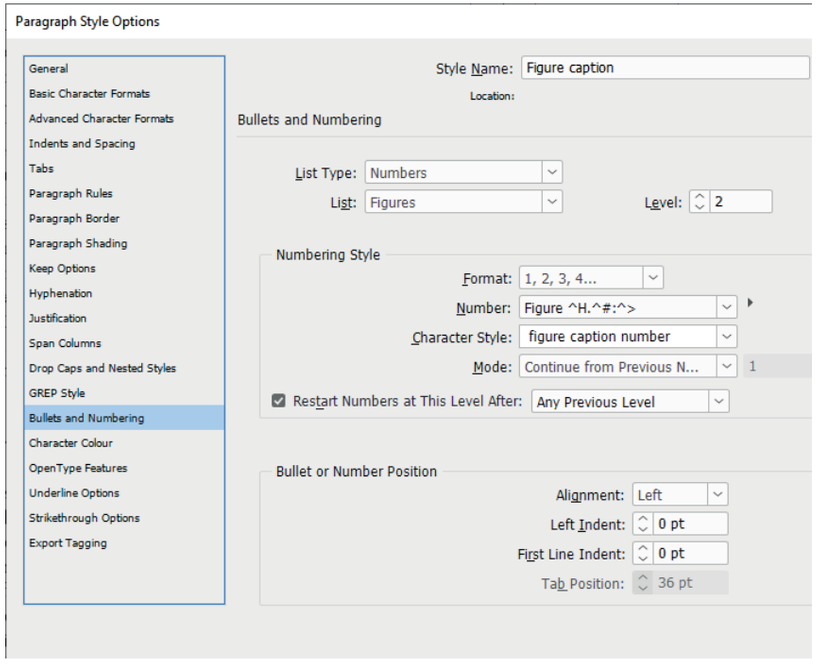

Figure 44. The caption style's settings

If there is no chapter title in the document, the number string is set to `^#:^>`, and the level to 1.

(To set the chapter number, open the Pages panel's fly-out (click the hamburger icon) or right-click the document's first page in the Pages panel, select `Numbering & Section Options` and set the chapter number in the `Start Chapter Number At` field.)

A numbering bug

The automatic numbering of the captions works fine, but there is a problem when two figures are placed on the same page: the figure numbers are the wrong way around, with the bottom figure's number higher than the top figure's number. This is an InDesign bug. It can be worked around with the script `021-caption-number-fix.jsx`.

The bug is pretty dumb. When numbered paragraphs are on separate pages, they're numbered correctly, InDesign follows the order of the pages. But when there are two or more numbered paragraphs (in separate frames, of course) then InDesign numbers them chronologically, that is, in the order in which they were created. Since the converter (for script-technical reasons) has to create the figures (and their captions) from the end of the document to the start, and two figures appear on the same page, the bottom figure was created before the top figure and its number will be lower than the top figure, not higher. This problem will occur also when you insert a figure later.

The only way to fix this that I'm aware of is to set each caption's start number. That way the captions are still numbered automatically (that's what InDesign thinks, anyway) and cross-references continue to work. This is what the script mentione above, `021-caption-number-fix.jsx`, does. I have used that script extensively while preparing this manual!

To set a start number manually, place the cursor in the paragraph whose number you want to set, open the Paragraph panel, click the hamburger menu in the panel's the top–right, and select Bullets and Numbering. In that window select Start At from the Restart Numbers dropdown, and enter the start number in the field to the right of the dropdown.

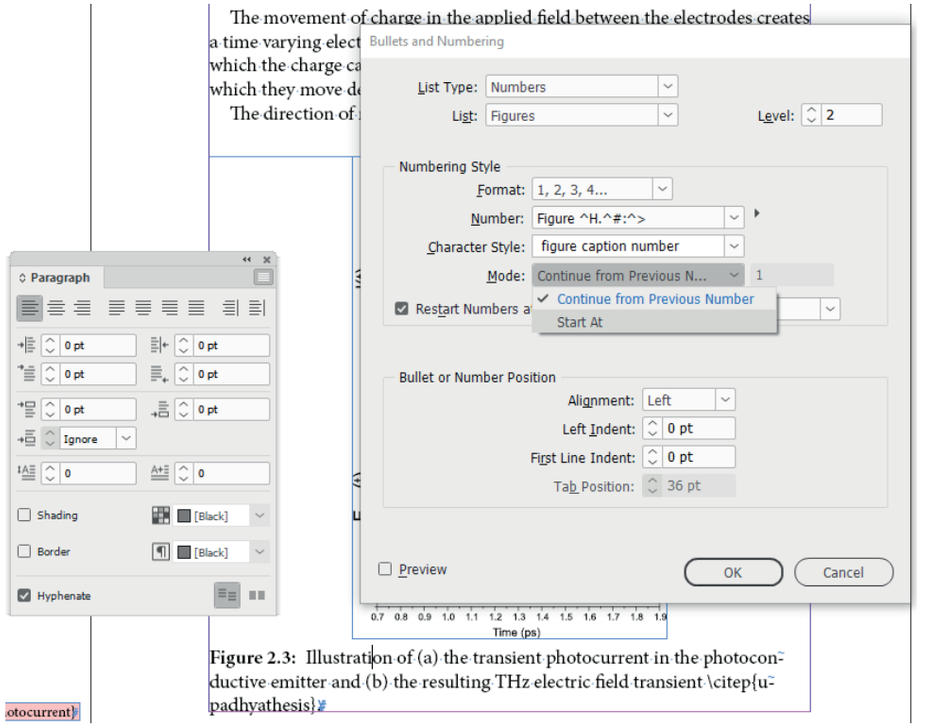

Figure 45. Setting a paragraph number

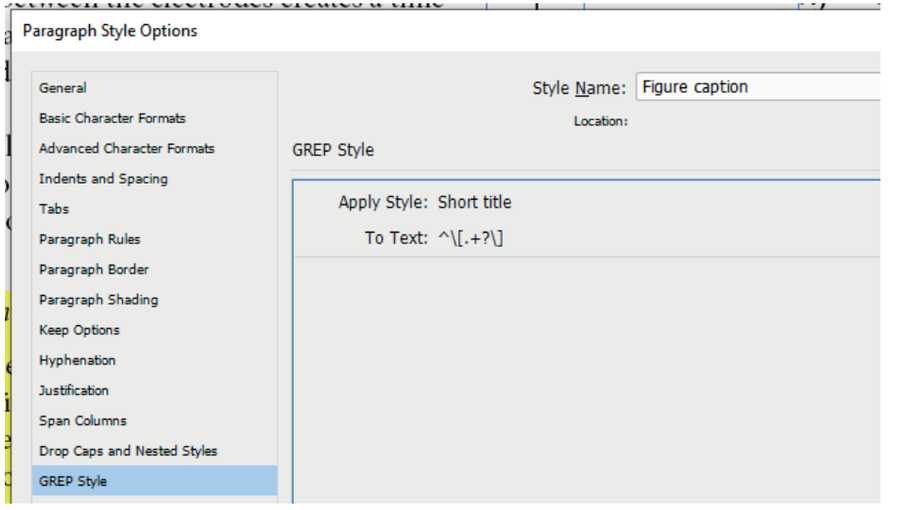

Figure 46. GREP style to hide short titles in captions

Short title

Captions can contain a short title for use in the table of contents. In Latex such a short title is placed in brackets at the start of the caption:

```
\caption[Band diagram]{Semiconductor band diagram
showing creation of an electron-hole pair by photo-
generation}
```

The converter leaves the short title in place and adds a character style (`Short title`) so that it is hidden in the caption. You can see it in the Story editor (Edit > Edit in Story Editor).

The character style is applied as a GREP style; it hides the text by setting it to 0.1 points, 1 per cent horizontal scale, superscript, and, for good measure, it applies the paper's colour.

The `Short title` style is applied as a GREP style because a 'normal', non-nested style causes problems in the table of contents.

Cross-references

Labels (cross-reference targets) and references (cross-reference sources) are placed in anchored text frames on the pasteboard. In Figure 47 you see the reference's frame pointing to where the reference will be inserted (later, with a separate script). The label in the caption, too, is anchored. You don't see the dashed line because the caption's story isn't selected, but you can see the anchor (the blue Yen-symbol) at the end of the caption.

Cross-references are resolved with a separate script because in a book job all chapters should be converted before you can set the cross-references; see page 49 for details.

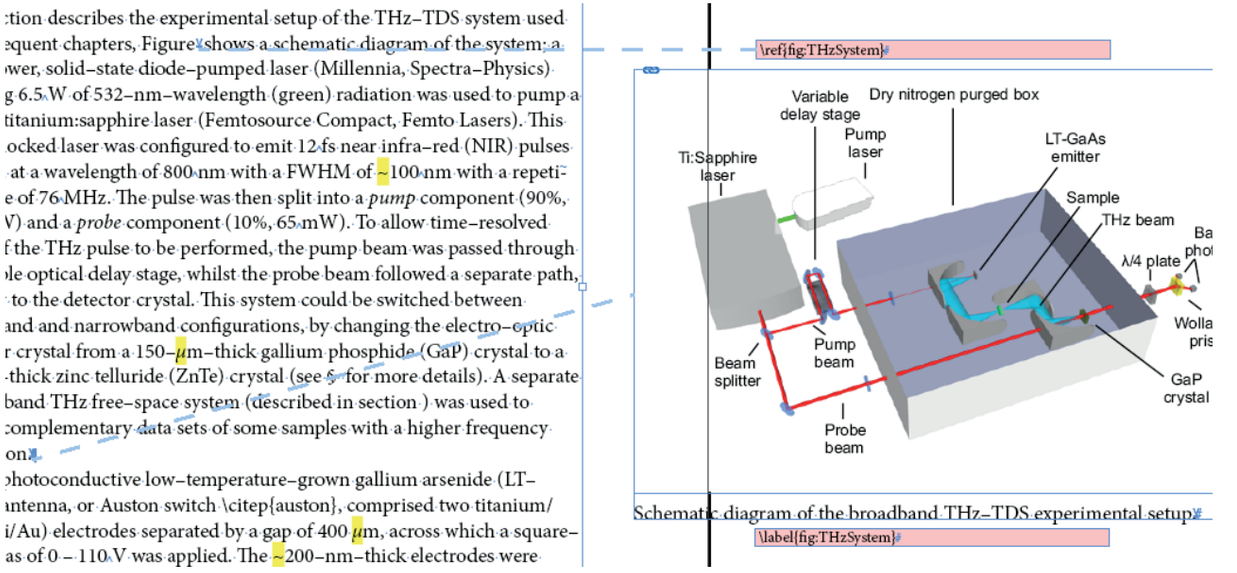

Figure 47. Cross-reference labels and references are in anchored frames

Tables

There are two types of table: in-text and floating. An in-text table sits in a paragraph that is part of the text stream and moves with the text like all other text. A floating table sits in a separate text frame, is almost always placed at the top or the bottom of the page, and stays there even when the text reflows.

In Latex an in-text table is defined in a `\tabular` environment. It is usually not numbered, and often doesn't have any rules. It's sometimes called 'text table'.

A floating table in Latex is simply a tabular environment embedded in a `\table` environment. Floating tables usually have a (numbered) caption.

Essentially it works the same in InDesign. An in-text table sits in a paragraph in the main text, probably using a dedicated paragraph style that (among other things) centres the table horizontally. To create a float, the paragraph containing the table is embedded in a text frame together with the caption. That frame is placed somewhere on a page.

(Note that floating tables and figures are similar: a separate text frame with a caption and an image or a table. And the way that they're placed and use cross-references, too, is similar.)

```
\begin{tabular}
...
\end{tabular}
```

```
\begin{table}
  \caption{...}
  \begin{tabular}
  ...
  \end{tabular}
\end{table}
```

Figure 48. Latex code for an inline (left) and a floating table (right)

Table formatting

The script converts Latex `\tabular` environments to InDesign tables. A table style is created on the fly and cell styles for the body rows and one each for the first and the last row. Paragraph styles are created for the cells in the top row, the body rows, and the bottom row.

Most formatting instructions in a Latex table are carried over to the InDesign table. Various types of rule are supported (`\hline`, `\cline`, `\toprule`, `\midrule`, `\bottomrule`). Top and tail rules are set to 0.5 points, internal rules to 0.25 points. This can be changed in the styles after the document has run, or in the inline-tables.jsx script. (It will be made configurable in some next version of the script.)

Kinetic data for the wild–type, and mutated strains of ECAO measured in the first set of THz–TDS experiments. Data used with thanks to Dr. M. A. Smith, University of Leeds.

ECAO strain	k_{cat} (s^{-1})	K_M (μM)	k_{cat}/K_M ($\mu M^{-1}s^{-1}$)
WT	15	1.2	12.5
E573Q	0.003	1.1	0.0027
I342F	4.1	1.6	2.6
I342F+E573Q	0.48	13.5	0.036

Figure 49. A formatted table

The alignment of columns is applied as defined in the table, and overrides are applied as well. @-expressions, when they insert some text, are applied, otherwise they are ignored, so tables that contain @-expressions usually need some post-processing to adjust column widths and/or cell insets, but content-wise everythig is carried over to InDesign.

Columns are snapped to their contents except when a column width is set in the Latex code (using the `p{...}` parameter).

InDesign's table formatting is a mix of settings applied to the table and to the text. For example, horizontal alignment in cells (and columns) is set as a paragraph attribute applied to the text. Vertical alignment in a cell is an attribute of the cell. And to centre a table horizontally, centring is applied to the paragraph that holds the text: horizontal alignment is not a table attribute.

Figure 50 shows an example of a table with various format instructions. The example is from Lamport's book (*Latex: A document preparation system*, p. 204). On the left is the Latex code, on the right, the script's output. Vertical rules weren't applied. And in the 'low' and 'high' columns the insets should be changed to bring the prices together so that they appear properly ranged. This is very fiddly and may be added to the script later.

```
\begin{tabular}{|r||r@{--}lp{1.25in}|}
\hline
\multicolumn{4}{|c|}{GG\&A Hoofed stock}
  \\ \hline\hline
&\multicolumn{2}{c|}{Price}& \\ \cline{2-3}
\multicolumn{1}{|c||}{Year}
& \multicolumn{1}{r@{\,\vline\,}}{low}
& high & \multicolumn{1}{c|}{Comments}
 \\ \hline
1971 & 97 & 245 & Bad year \\ \hline
  72 & 245 & 245  & Light trading due to a
                   heavy winter \\ \hline
  73 &   245 & 2001 & No gnus was very good
            gnus this year \\ \hline
\end{tabular}
```

GG&A Hoofed stock			
	Price		
Year	low	high	Comments
1971	97–	245	Bad year.
72	245–	245	Light trading due to a heavy winter.
73	245–	2001	No gnus was very good gnus this year.

Figure 50. One of L. Lamport's example tables. Original Latex code on the left, script output on the right (vertical rules not applied). Column insets to be set manually to bring together the figures in the 'low' and 'high' columns

Floats

Like figures, floating tables are placed as custom anchored text frames on the pasteboard. The anchors are placed at the location of the Latex table code. A table is created in a text frame together with its caption, and if there's a cross-reference label it's processed as usual.

The script creates an object style for the table container on the fly. Its name is `Table` and can be tweaked if necessary, for example, to change the space between the table and the caption. The table itself is created as described above.

The table's container frame is set to the width of the text.

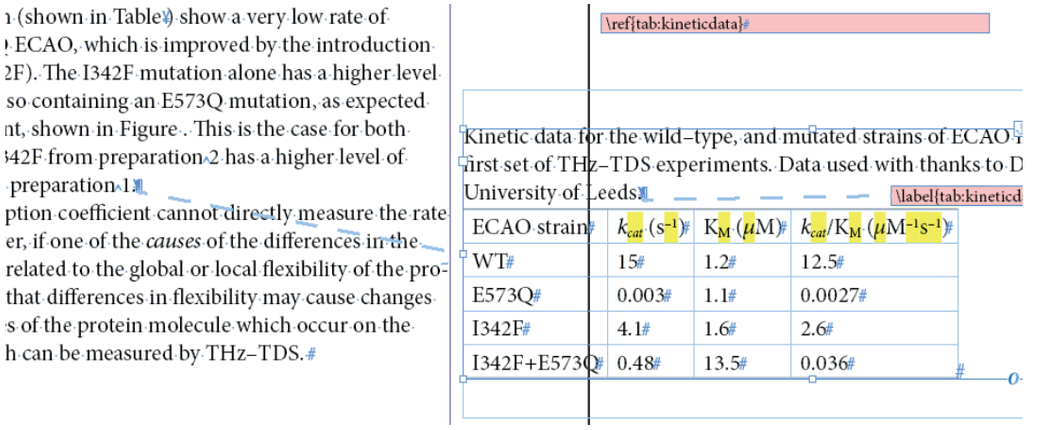

Figure 51. A floating table initially placed on the pasteboard as an anchored frame (the heavy vertical line is the page edge)

Captions

Tables captions are handled in exactly the same way as figure captions. So are short-title items. See pages 35–36 for details.

Cross-references

Labels and cross-references are treated as described for figures.

Footnotes

Table notes are placed immediately after the table because the table is in a separate text frame, in other words, in its own story. Unfortunately, InDesign allows just one numbering scheme per document, so the table notes will be numbered with Arabic numbers. You will probably want to change that to roman numbers, for which you can use a script, table footnotes. See also the section on footnotes, page 47.

Arrays

An array is basically a table without rules. The difference between arrays and tables is substantial in Latex (it has to do with math mode versus text mode and the differences that follow from that), but for InDesign the difference doesn't matter.

Lists

The three list-making environments `enumerate`, `itemize`, and `description` are produced as expected.

Enumerate

A paragraph style (`enumerate`) and an InDesign list definition (`enumerate`) are created on the fly. The paragraph style is based on the `Body (no indent)` style. The paragraph's left and first-line indent are set together with a tab stop to create hanging indents.

The numbering mode is set to 'Continue from previous number' in the style; the script restarts numbering every list by setting the first list item's start number to 1.

The paragraph style's space before and after are set to 6 points; change that as necessary. The list's internal spacing is zero by setting 'Space between paragraphs using the same style' to 0.

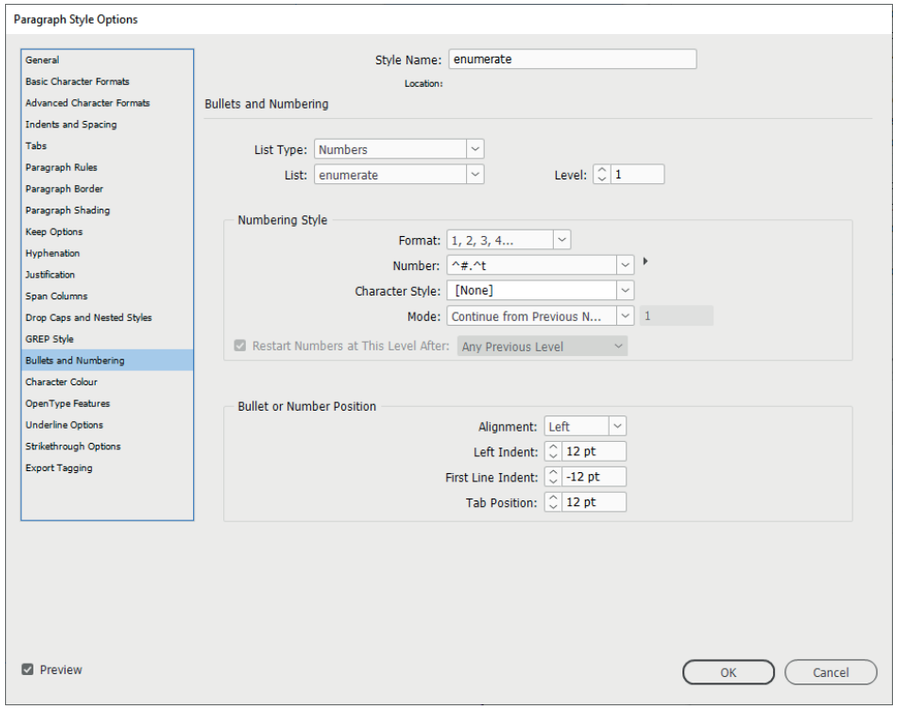

Figure 52. Paragraph style for numbered lists (enumerate)

Itemize

Itemized lists are formatted as bulleted lists. The script uses the standard bullet character, which can be changed in the paragraph style.

Other paragraph attributes (spacing, hanging indents) are the same as those of the `enumerate` style.

Description

The script creates a paragraph style (`description`) and a character style for the labels (`description label`), which applies bold. The character style is applied as a nested style in the paragraph style. The blue backslash in Figure 53 following the labels is InDesign's end-nested-style-here code, which sets the scope of the nested character style.

The space after the label is an em-space (the blue horizontal bar with thee dot below it), which is hard-coded in the script. It's possible to implement a style-driven space, but that's pretty convoluted and since

gnat\ A small animal¶
gnu\ A big animal¶

Figure 53. Description

the space is almost always either an en- or en am-space, if it should be an en-space it's an easy find-and-replace action. (If I ever get round to using a configuration file, the space after the description label will be part of it.)

Tabbing

Latex's `tabbing` environment is rendered as a sequence of tabbed paragraphs. A paragraph style is created (`tabbing`) which sets 6 points space before and below the tabbed list.

The commands `\pushtabs`, `\poptabs`, and `\kill` are supported. `\+` and `\-` are not (yet).

Margin paragraphs

Margin paragraphs (`\marginpar{...}`) are placed in anchored text frames. The script creates an object style for the frame (`Margin note`) and a paragraph style (Margin note) for the content. The width of the frame is set to 72 points. To change this, change the object style.

The frame is centred vertically relative to its anchor. To show the anchor, enable Normal screen mode (View > Screen Mode > Normal) and enable the display of text threads (View > Extras > Show Text Threads). The anchor is shown as a blue Yen-symbol; a blue dashed line connects the anchor and the text frame. To change the vertical alignment relative to the anchor, change the object style.

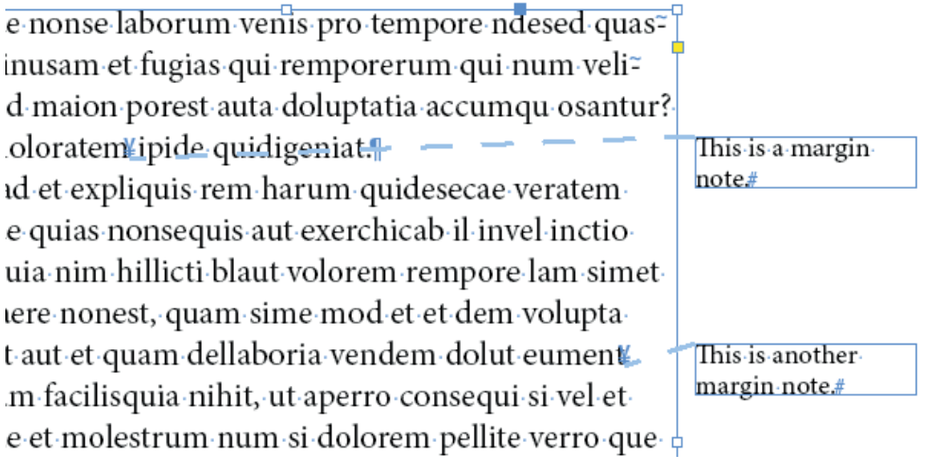

Figure 54. Margin paragraphs

Quotations

For each type of quotation – `\quote` and `\quotation` – a paragraph style is created on the fly. They are based on `Body (no indent)`. In both styles space before and after is set to 6 points, internal paragraph spacing is zero.

Footnotes

InDesign's footnotes are considerably less flexible than Latex footnotes. In InDesign you can set the start number only at the level of the document. You can have only one numbering style per document. Numbering restarts can be set only at the level of the spread, section, or page. And, finally, footnote numbering restarts at every story. For this reason any parameters in a footnote command are ignored.

Style parameters (footnote options)

Style parameters (separation rule, space between footnotes and text, etc.) are ignored, you set all these in the Layout tab of the Footnote Options window (Type > Document Footnote Options > Layout).

Some style parameters can be set at the page, but because the InDesign document is bound to flow differently from the generated Latex document there's no point in adding these Latex parameters as local overrides in InDesign.

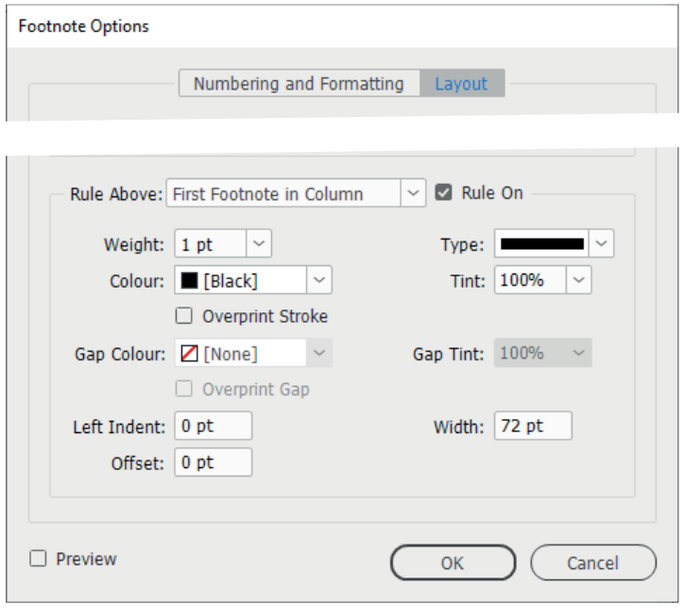

Figure 55. Footnote options

Footnote styling

The script creates a paragraph style for the footnotes (`Footnote`), based on the `Body` style, and a character style for footnote references (`Footnote reference`).

The separator (that is, the footnote number and text separator – `^t` for Tab by default) can't be styled directly in this window. The simple solution is to create a nested style or a GREP style in the footnote paragraph style.

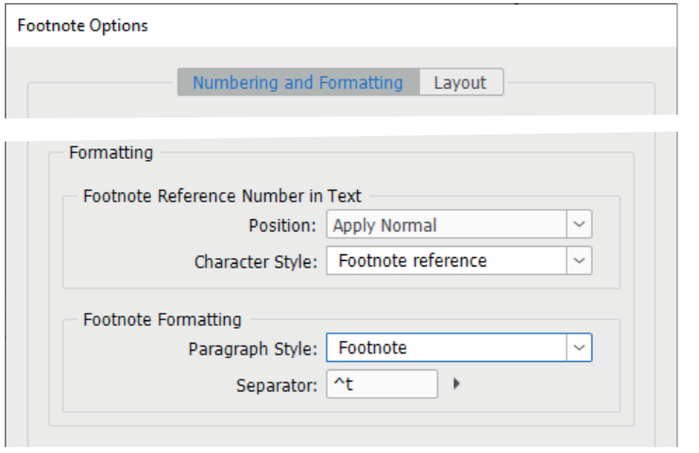

Figure 56. More footnote options

Table notes

Tragically, Adobe decided to do table notes the way MS Word does them. If a table is in the text flow (an in-text table), then its notes are part of the main text's footnote stream. Table notes are numbered as part of the main text's notes, and they appear at the foot of the page rather than below the table.

Because footnotes are a property of the story, this isn't a problem. The script converts tables as floats, which form their own story, and therefore they are placed below the table and their numbering starts at 1.

However, footnote numbering style can be set only at the level of the document, so table notes are numbered Arabic, just like the notes in the main text. To change this a separate script is needed, see table footnotes.

Footnotemarks

InDesign doesn't have the equivalent of Latex's `\footnotemark` and `\footnotetext` commands so they are converted to footnotes. In the InDesign document they'll form part of the document's footnote stream.

Cross-references

In InDesign it's not possible to create cross-references to footnotes. The `\label` and `\ref` markers stay in place unchanged so you can replace them with hard numbers.

If live cross-references are a must you can use DTP Tools's plug-in. The plug-in is fully scriptable and works well. An additional interesting feature provided by DTP Tools is its ability to disable cross-reference updating.

Cross-references

The converter moves Latex's references and labels (in InDesign-speak that's cross-reference sources and destinations) to anchored text frames on the pasteboard (Figure 57). The dashed rules indicate where the original `\ref` and `\label` codes were in the text (if you don't see those lines, enable Normal screen mode (View > Screen Mode > Normal) and enable the display of text threads (View > Extras > Show Text Threads).

In book jobs you'd resolve the cross-references only after all chapters have been converted. Open the book file and run the script `030-cross-references.jsx` from InDesign's Scripts panel.

The result is shown and discussed on the following page.

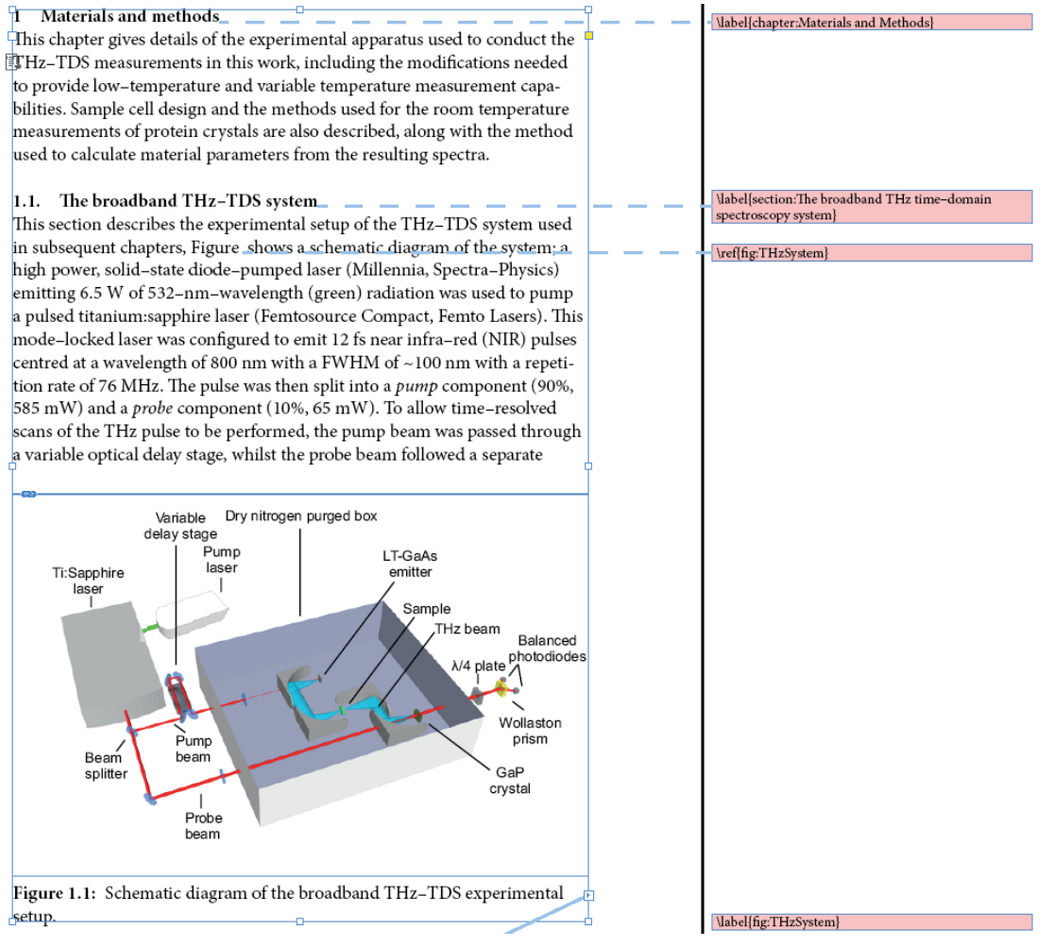

Figure 57. Cross-references are placed as anchored text frames

The created cross-references are displayed in the Cross-References panel (see Figure 58). The names are those of the targets. In this panel those names indicate the cross-reference types: `(1.2)` is an equation; `1.1.1` is a section number; the rest, of course, are the sources of figures and tables.

The numbers on the right-hand side of the panel are page numbers of the sources, the dots represent the targets (green dots for valid cross-references; `PB` stands for 'pasteboard'; invalid/out-of-date cross-references are indicated by yellow triangles). Both the numbers and the dots can be clicked to jump to their locations in the document.

The anchored frames with references and labels always start out shaded light red. The cross-reference script removes the shading from the frames whose content could be resolved. In Figure 58 you can see that the reference to Figure 1.1 was resolved (second line in section 1.1).

The sample file shown in the figure is part of a book and I ran the script without the book file or any other documents open. The cross-reference frames that remain red are those that could not be resolved: their references are in a different file.

The frames can be removed because after fixing the cross-references they don't serve any purpose – though they could stay there since they won't appear in exported PDFs or in print.

Footnotes

In InDesign it's not possible to create cross-references to footnotes. If you really need them you can use DTP Tools.

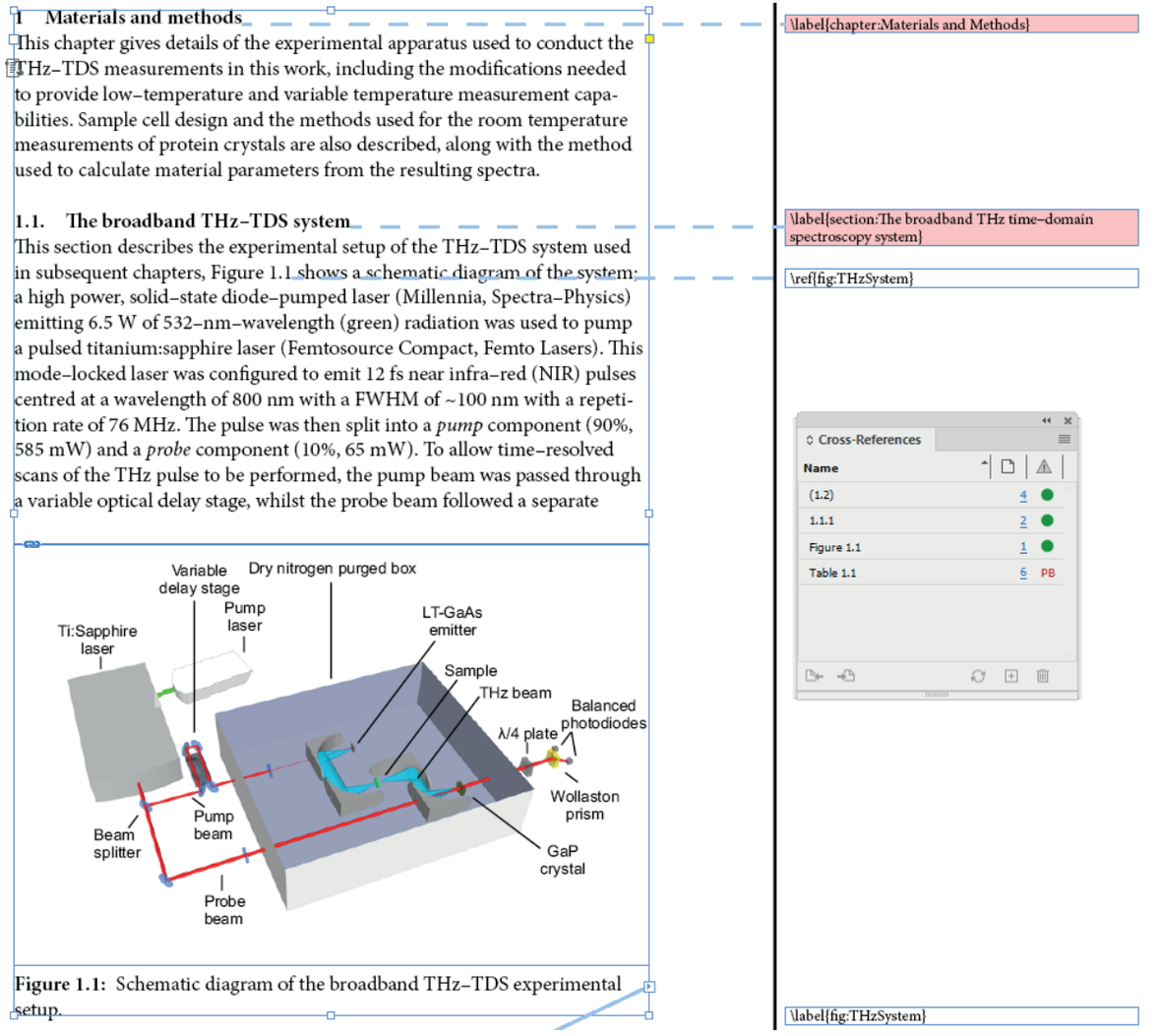

Figure 58. Resolved cross-references

Hyphenation exceptions

Latex hyphenation exceptions are coded as `\hyphenation{...}` in the preamble or in the document file. The command's argument is one or more words with possible break points indicated by dashes.

Hyphenation exceptions can be handled in two ways in InDesign. You can add them to the exception dictionary or you can add discretionary hyphens in the document. I always add common exceptions in the dictionary and rare, idiosyncratic, words in the document itself because I don't want to add lots of very unusual words to the dictionary.

Latex, on the other hand, doesn't offer that choice and always adds possible word breaks in the document.

The script lets you choose where exceptions should be applied: in the dictionary or in the document. It adds exceptions from the preamble in the dictionary, and the exceptions found in the document are marked up in the document by adding discretionary hyphens.

To use the option to add words to the exception dictionary, make sure that the correct language is applied in the `Body` paragraph style. The script needs the language name so that it adds the exceptions to the correct dictionary.

To set the paragraph style's language, edit the style and go to the Advanced Character Formats tab and select the language from the Language dropdown.

```
\hyphenation{ac-tiv-ity}
\hyphenation{abil-ity abil-ities}
```

Figure 59. Hyphenation commands in the preamble

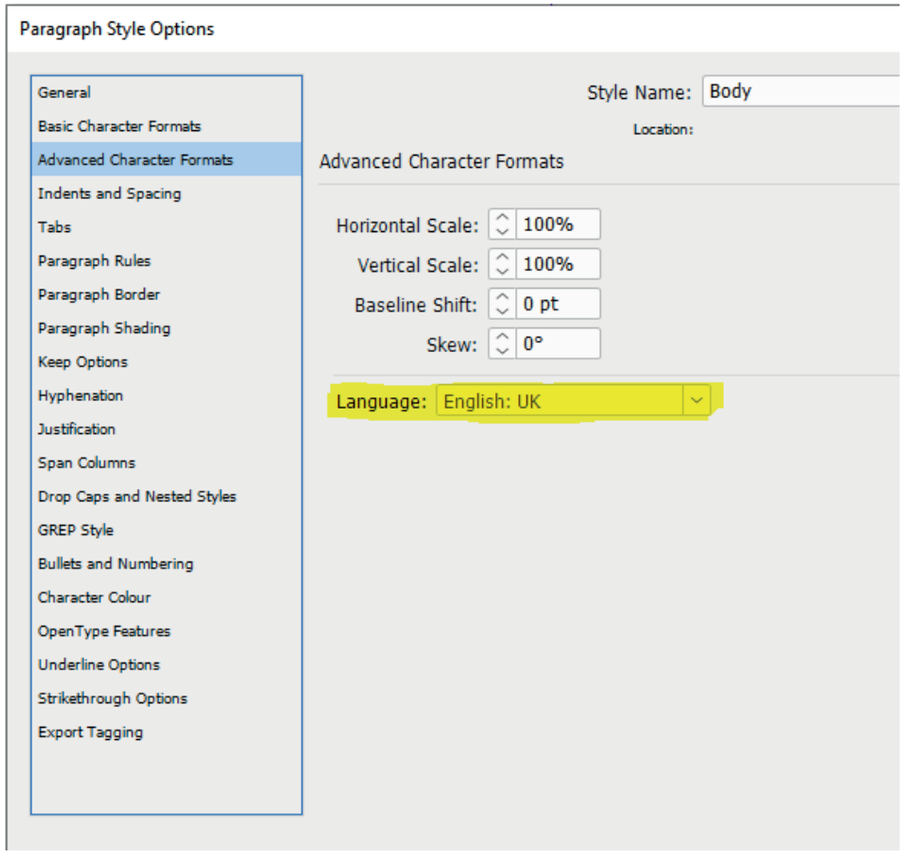

Figure 60. Setting the document's language

Citations and bibliography

Citations in the text aren't processed by the converter, they're handled by a separate script (`citations-and-bibliography.jsx`) that should be run after the converter.

The script doesn't handle Latex's standard citation command (`\cite{...}`). Instead it supports the citation format of the `natbib` package as described in the documentation of the natbib package (Patrick Daly, *Natural Sciences Citations and References*). This package has been part of the standard Latex installations for many years and is the *de facto* standard.

For text citations, the package has two commands, `\citep{...}` and `\citet{...}`, for parenthetical and text cirations, respectively: *(Jones 2025)* and *Jones (2025)*. Daly states that Latex's `\cite{...}` command behaves like `\citet` for author–year citations but like `\citep` for numerical ones. The use of `\cite` is therefore discouraged. If your text contains `\cite` commands, simply change them to either of the natbib versions.

Of the commands described by Daly, the following are supported: `\citet`, `\citep`, `\citealt`, `\citelalp`, `\citenum`, and `\citetext`. The remaining commands may be supported in a future release of the script.

Of the package parameters, the following three are supported: `number`, `sort`, and `sort&compress`. The script tries to read the parameters from `\usepackage` command, which should be in the preamble, which should be in the open InDesign document's folder. The command should be in the standard Latex format, e.g.

```
\usepackage[numbers, sort&compress]{natbib}
```

If there's any problem processing the preamble (file not found, usepackage not found, etc.), the script defaults to `numbers == false`.

The `sort` parameter is defined only for the number format (i.e. the citations have bracketed numbers instead of years and entries in the bibliography are preceded by corresponding bracketed numbers). When the `number` parameter is not present and `sort` or `sort&compress` is,

```
%fixed

@article{
abobakr
title=Brilliant, Coherent Far-Infrared (THz) Synchrotron Radiation
author=M. Abo-Bakr \emph{et al.}
journal=Physical Review Letters
volume=90
number=9
pages=094801
numpages=4
year=2003
publisher=American Physical Society
}

@article{
Adam
author=A. J. L. Adam and P. C. M. Planken and S. Meloni and J. Dik
journal=Optics Express
number=5
pages=3407–3416
publisher=OSA
title=Terahertz imaging of hidden paint layers on canvas
volume=17
year=2009
}

@article{
alben
author=Alben, J. O. and Caughey, W. S.
title=Infrared study of bound carbon monoxide in the human red blood cell, isolated hemoglobin and heme carbonyls
journal=Biochemistry
volume=7
number=1
pages=175–183
year=1968
}
```

Figure 61. A .bib file

then the script lists multiple citations chronologically, as in (Smith 2010, Jones 2012), irrespective of the order in which they appear in the `\citet` command.

The bib file

Naturally, the citations can be processed, and the bibliography generated, only if the `.bib` file – the bibliography database – is present. This is the only file that the script can work with to process citations and the bibliography.

The bibliography style

The script supports just one bibliography style, `unsrtnat`, but this can be used for numerical and author–year mode. If the `\usepackage` command includes the `numbers` parameter, the bibliography is organised chronologically (that is, in order of text occurrence) with bracketed numbers preceding the entries. If the `numbers` parameter is absent, the bibliography is created APA style, without bracketed numbers and sorted alphabetically.

The `unsrtnat` style is the only one supported by the script, it's defined in `unsrtnat.jsx`, which is the equivalent of Latex a .bst file. It lives in the BST subdirectory. This directory also contains two other style scripts (plainnat.jsx and abbrvnat.jsx), but they are the same as unsrtnat.jsx. You can use unsrtnat.jsx as an example to define other styles.

Hyperlinks

References (both numeric and full text) can be linked to the bibliography using a hyperlink. This hasn't been implemented yet. A better solution is in fact pop-up references: roll over a reference with the mouse and a pop-up note with the full reference appears at the foot of the page.

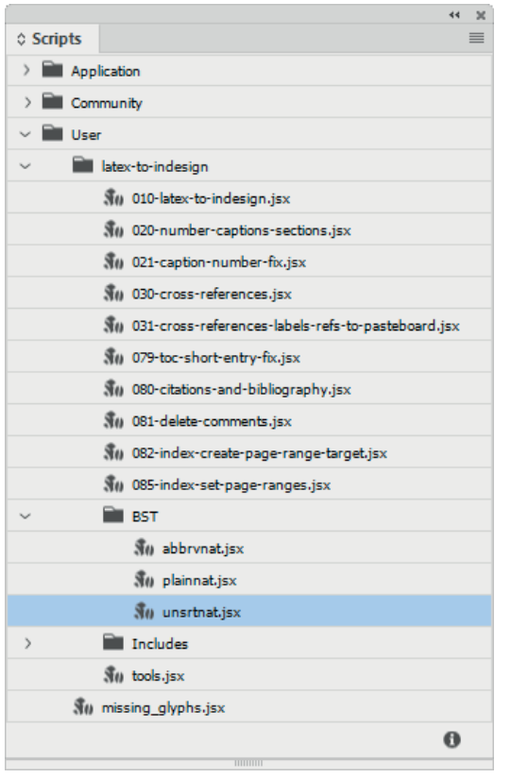

Figure 62. The BST folder

Table of contents

The table of contents and lists of figures and/or tables are handled by InDesign in the usual way. However, Latex has two features that InDesign lacks, but which can be implemented in InDesign: short titles and additional text for the TOC that doesn't occur in the body of the text. In what follows we'll first outline the standard TOC, then we'll outline how these two Latex features can be handled.

The standard TOC

The converter creates styles for the various headings it finds in the text (part, chapter, section, subsection, and subsubsection) and things like table and figure captions. You find these styles in the Paragraph Styles panel.

To generate a table of contents, create a table-of-contents style as usual. For each item that should appear in the TOC, create a new paragraph style – for example, create a `section toc` paragraph style along the `section` style used in the text. Set up the TOC style as shown in Figure 63. In a book, don't forget to check the Include Book Documents checkbox.

Additional TOC text

This is one of the two features that InDesign lacks. It's sometimes useful to include a heading in the TOC that makes sense only in the TOC and therefore doesn't appear in the main body of the text. The Latex command for including such a heading is `\addcontentsline`:

```
\addcontentsline{toc}{section}{English}
```

The command has three parameters. The first one determines in which TOC the text should be included: the main contents (`toc`), the list of figures (`lof`), or the list of tables (`lot`). The second parameter determines the appearance of the additional text. In the case of the above example the additional text is formatted as a section entry. The third parameter is the text to be inserted in the table of contents.

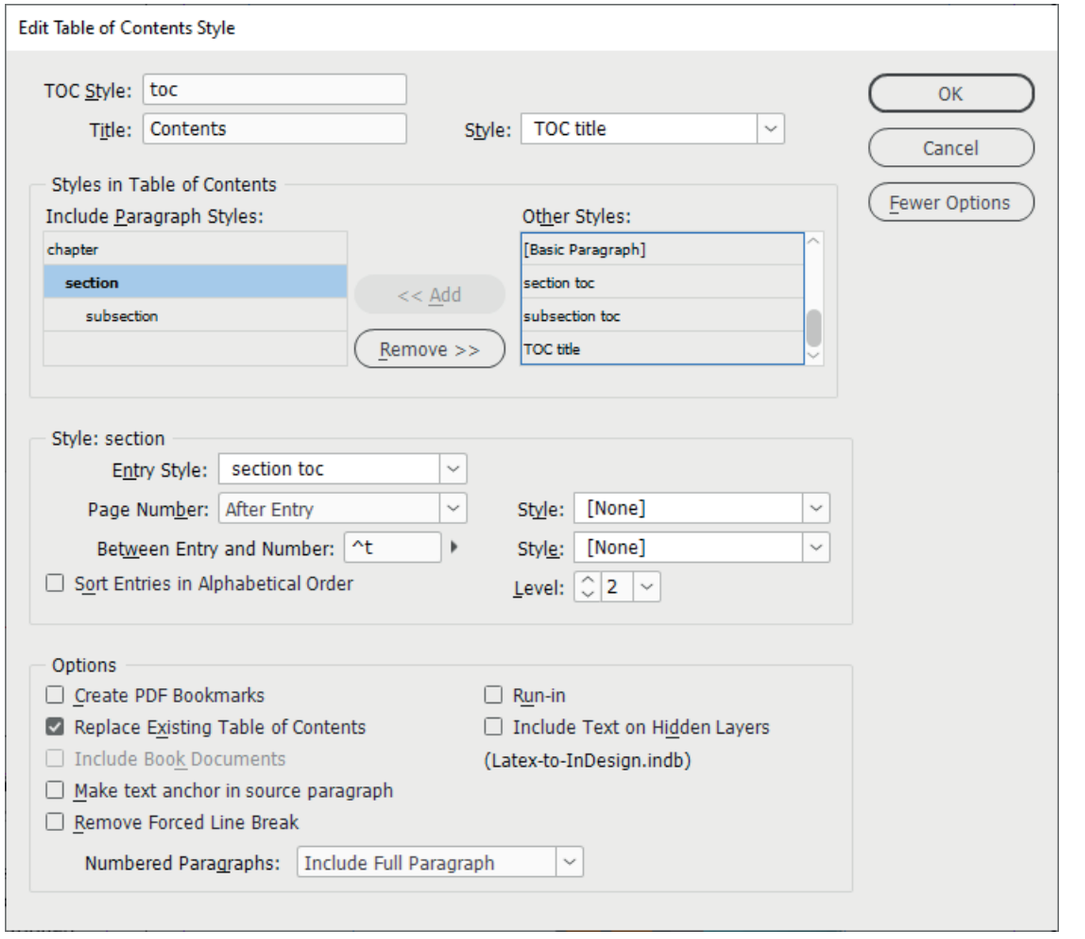

Figure 63. A TOC style

In the text these `addcontentsline` items are hidden but you can see them in the story editor. Only the text of the entry remains (here, *English*). The converter created a paragraph style whose name is constructed from the first and the second parameters of the command: `section (dummy TOC) toc`. The text is hidden in the usual way by applying 0.1 point size, 0 leading, and 1 per cent horizontal and vertical scale.

Note: Because point size can't be less than 0.1 points these `addcontentsline` paragraphs add 0.1 point vertical space.This would be a problem only if the next line of text should be on the grid and if you align text to the baseline grid. If this situation occurs the remedy is to reduce the next line's leading by 0.1 point. If you don't align the text to the baseline grid then don't bother.

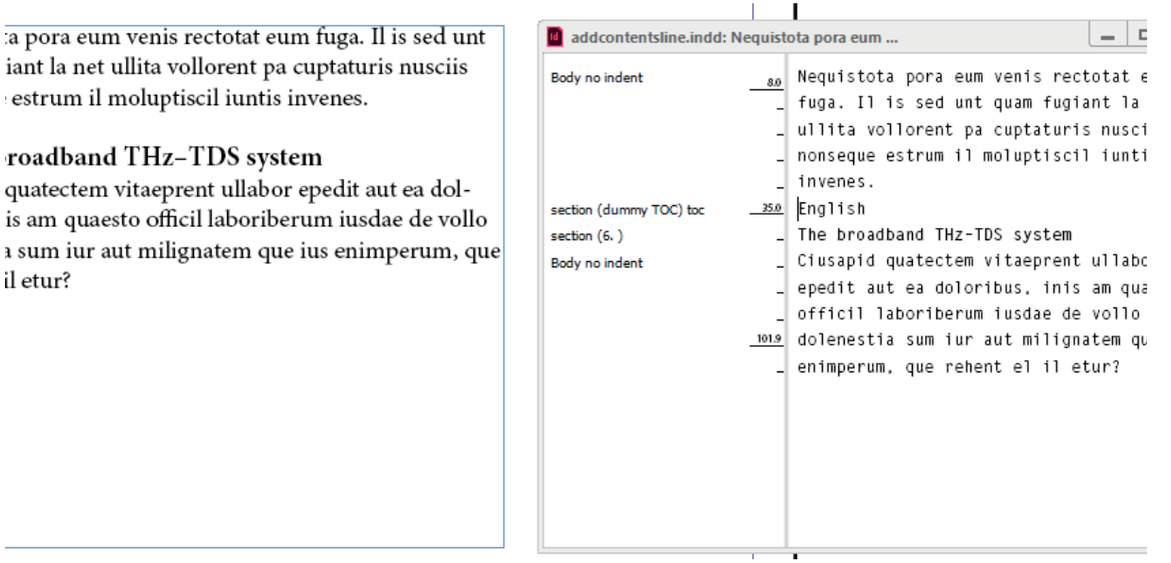

Figure 64. Additional table-of-contents text

Short titles

The second Latex feature that can be replicated in InDesign is the shortened heading or caption. The length of a running header should generally not be longer than about two-thirds of the width of the text area, so long headings should be shortened to fit. Tables and especially figures often have long captions, and these should be shortened in the table of contents. Short titles are given as a parameter of the heading or caption command:

```
\caption[Short title]{Long title}
```

Long title appears in the text, *Short title* is used in the TOC and in running headers. Adding this functionality isn't entirely straightforward but can be done. We'll start with the table of contents.

Short titles in the table of contents

The converter deals with short TOC titles as follows. A character style `Short title` is created and applied to the short title so that it's hidden (0.1 point type size, 1% horizontal scale).

Take the caption in Figure 66. A caption like this will end up in the converted document as shown in Figure 65. The story editor (Edit > Edit in Story Editor) shows the short title (it's in brackets), while the short title isn't visible in the document's window. Select anything in the short title and you'll see that the character style `Short title` was applied by the GREP style (circled red).

When you generate the table of contents the full caption will be included, that is, the short title and the display caption (Figure 67).

All that's then left to do is to delete the long title and the brackets that wrap the short title. Alternatively, set two GREP styles in the TOC style that hide the first bracket and everything from the closing bracket to the tab or right-indent. You can use the same `Short title` character style, see Figure 68.

(The second GREP expression is `\[.+?\K\].+?(?=[\t~y])` – in case you want to use it you can copy it from here.)

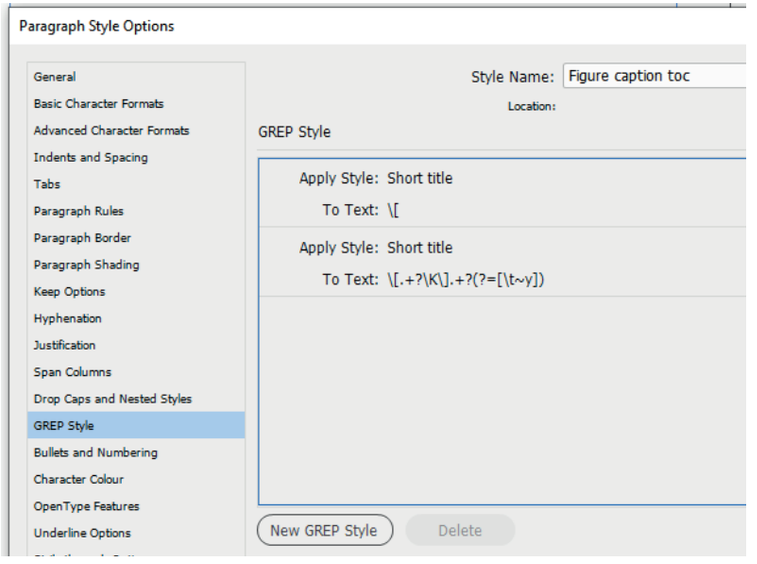

Figure 68. Hiding the long title in the table of contents

```
\caption[Photocurrent and resulting THz electric field]Illustration of (a) the
transient photocurrent in the photoconductive emitter and (b) the resulting THz
electric field transient}
```

Figure 65. A caption with a short title; Latex code

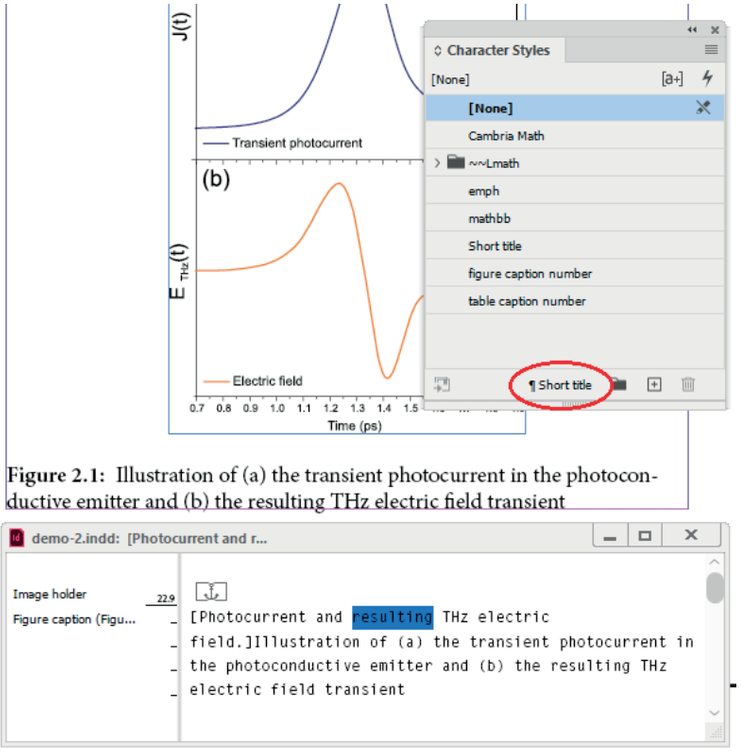

Figure 66. A caption with a short title in the document and the story editor

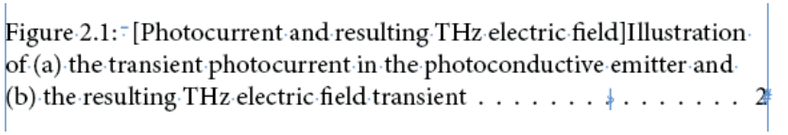

Figure 67. Caption with short title in the list of figures

Short titles in running headers

Short titles in chapter and section headings are used in running headers as well. You can set these up using InDesign's text variables in the usual way.

However, there may be a catch here. InDesign doesn't include any chapter or paragraph numbers in the running headers, which is generally correct in my experience, but sometimes the paragraph numbers must be included.

Another 'feature' of InDesign's running head text variables is that internal formatting isn't honoured, so if you use italics, for instance, in a section heading, the italics won't appear in the running header.

For these reasons the most appropriate approach is to use a script that places the running headers on rectos as text rather than using a variable. This is a general InDesign issue, for a more detailed description and scripts, check these two links:

running-headers.html
text-variables-in-running-headers.html

Comments

Comments – text between a single unescaped percentage symbol and a return character – are placed as anchored text frames. The percentage symbol is kept; the frame is light blue.

Consecutive comments are placed together in a single frame. It's possible that a big comment frame masks a smaller one.

The script creates an object style for the frame (`Comment`) and a paragraph style (`Comment`) for the content. The width of the frame is set to 64 mm. All this can be changed in the styles.

The frame is placed where the comment used to be in the text. To show the frame's anchor, enable Normal screen mode (View > Screen Mode > Normal) and enable the display of text threads (View > Extras > Show Text Threads). The anchor is shown as a blue Yen-symbol, a blue dashed line connects the anchor and the text frame.

To delete a comment, select it and press Del or backspace. To delete all comments you can look for the object style using the Find Object tab in the Find/Change window.

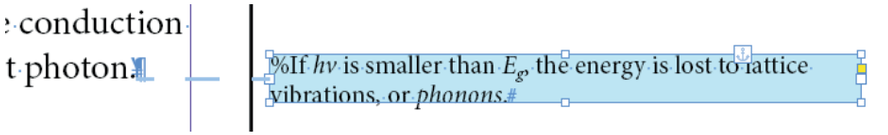

Figure 69. A comment

Index

InDesign's index feature is notoriously under-specified, but with some additional scripting the flexibility of Latex index markers can be successfully converted, and InDesign's index becomes quite powerful. Areas of concern are the styling of topic names, page ranges, and contextual awareness such as index markers in footnotes.

Topic styling

InDesign topic names can't be styled directly, you need an approach which is in fact much like Latex's. In Latex a topic name can include a text tag to indicate some character style, while the separate sort-order field can be set to ignore those tags and ensure that the topic is sorted correctly. For instance, in this Latex index item:

```
\index{alps@\textbf{alps}}
```

the topic is `\textbf{alps}`, and is preceded by the sort-order string (`alps`; the @-symbol separates the sort order and the topic name). This Latex index marker is converted to the InDesign page reference seen in Figure 70.

The topic will appear in the generated index as `\textbf{alps}`. A separate script is needed to remove the textual tag and apply the character style `textbf`.

There is a scriptless alternative using a nested GREP style, which involves hiding the textual tags while at the same time applying the character style.

To hide any text in InDesign, create a character style that sets the type size to 0.1 points and horizontal scale to 1 per cent. That's usually enough, but for good measure you could also apply superscript and page colour `Paper`. You would then use that character style in a nested GREP style as shown in Figure 71. The character style `textbf` is applied to text wrapped in braces, the character style `hide` to everything up to and including the opening brace and to the paragraph-final closing brace.

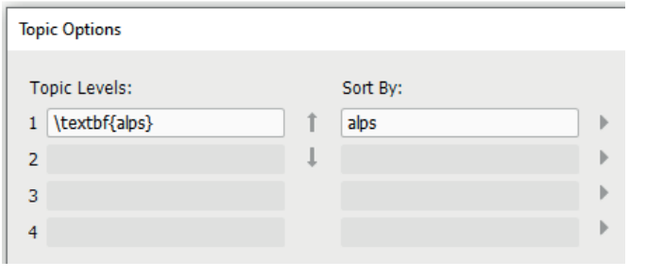

Figure 70. A page reference

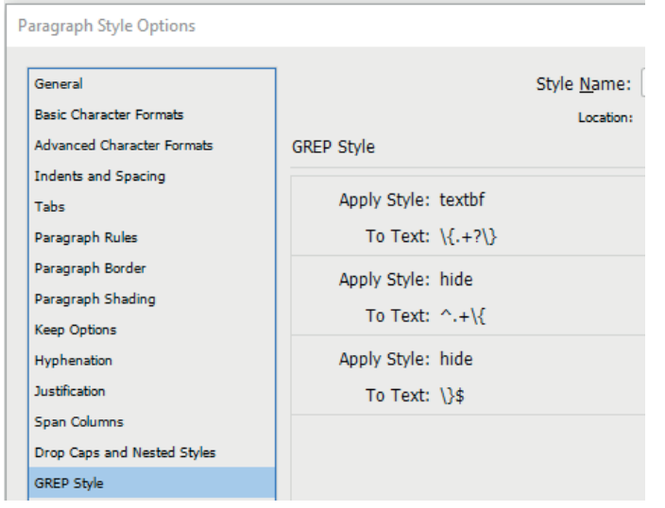

Figure 71. Defining a GREP style

The second approach is more convenient in that it's a set–up-and-forget feature, but it does involve some overhead and it probably doesn't do much for accessibility.

Page ranges

InDesign offers several ways to set up page ranges, most of which are useful, but it lacks the single most useful page-range approach, namely, an end-of-range marker as used by virtually all word processors and typesetters. But it's not too hard to script those markers.

In a Latex marker such as `\index{restivity|(}`, the opening parenthesis `(` indicates that it is the start of a range. The converter places a standard InDesign page reference there. Its type is set to `For Next No of Pages` and its range is set to 1 page. Later this type is used by the script that applies the page ranges by the time the index is generated.

The end marker, `\index{restivity|)}`, is placed as a graphic line, its name is set to `Range target:` followed by the topic name (the name is shown in the Layers panel). The graphic line has no width and is 6 points tall. The script applies an object style (`Range target`, created on the fly) to the line so that it (and all other end-of-range markers) can be made visible, which makes them easier to manage. Simply change the object style's stroke weight to 1 point and apply some colour. (This can cause some reflow of the text but when you change the style's stroke weight back to 0 points the text will flow back.)

If the topic is a subtopic the marker's name is constructed from the parent topic and the topic separated by `##`. For example, the end-of-range marker of the topic *whiskey, Islay* is labelled `Range target:whiskey##Islay`; see Figure 73.

Before you generate the index you need to run a separate script which sets the correct page count at the start-of-range markers. When the index needs to be regenerated you'll have to run the script again to re-set the page ranges.

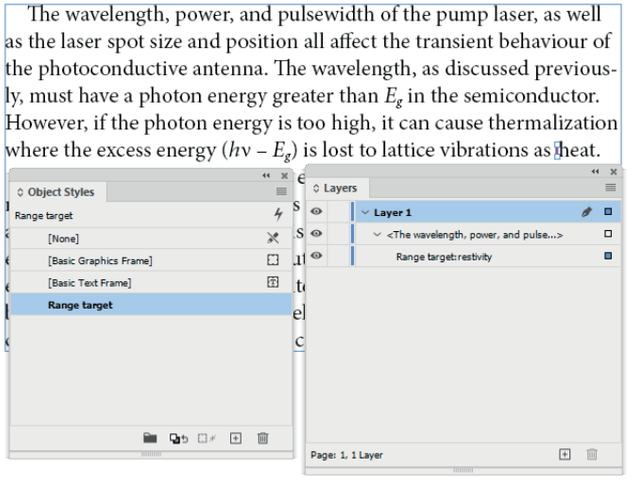

Figure 72. An end-of-range marker

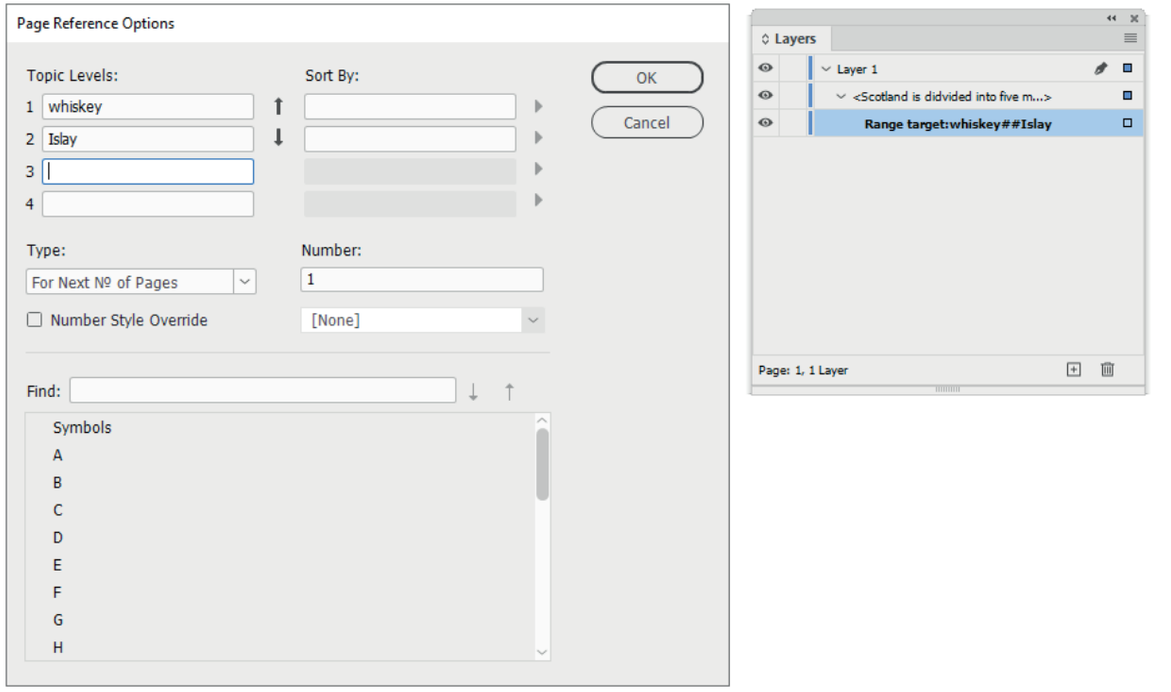

Figure 73. An end-of-range marker for a subtopic

Index references in secondary text

InDesign's index markers aren't aware of their context. It's therefore not possible to indicate that an index reference is in a footnote, a table, or anywhere else. This can be achieved by some scripts but they're not included here, they're not trivial.

Applying page ranges

By the time you're ready to generate the index, you'll first need to run the script that actualises the page ranges in the Index panel. The script is 070-index-set-page-ranges.jsx.

Script directory

The converter, Latex-to-InDesign.jsx, is a script that calls a number of include files. All these include files are in the `Include` folder which was created when you installed the scripts. Most of them can be run independently when you uncomment the function's call in the include file. Thus, the include file `figures.jsx` starts with these two lines:

```
//~ placeFigures()
function placeFigures () {
```

To run this file as an independent script, uncomment the first line and run the script from the ESTK or VSCode or save it (using a different name) and run it from the Scripts panel.

Apart from the include files called by the converter there are several scripts that should be run after the conversion of a Latex file. All these files are listed here together with a short explanation of what they do.

The order in which the include files are called in `Latex-to-InDesign.jsx` should not be changed.

Files run by the converter

first-things-first.jsx

- If the document in which you're placing the Latex file doesn't contain the paragraph style `Body`, the script creates it. The Body style is used as the basis for all other styles that are created by the script. The style is set to use Minion Pro at 10.5/13 points.
- All spurious white space (leading, trailing, sequences) is deleted.
- Various commands are removed that have no meaning for InDesign or that make no sense until you lay out the document (`\sloppy`, `\newpage`, etc.).
- The notation for math environments is normalised. There are various ways to code equations in Latex. The converter normalises them so that the math converter has less to do.

In-text formulas can be coded as `$...$`, `\(...\)`, and `\begin{math}...\end{math}`. The script replaces the latter two with the first.

Unnumbered display equations can be marked up by `\[...\]`, `$$...$$`, or `\begin{displaymath}...\end{displaymath}`. The script changes the first two formats to the last one.

Numbered display equations can be wrapped in just one code, so they don't need any normalisation.

– The text is cleaned up (remove leading and trailing spaces, etc.)

newcommand.jsx
The script looks for `\newcommand`, `\renewcommand`, `\providecommand`. and `\def`. It first looks in the active document, then it looks for a file `preamble.tex` in the active document's directory. These commands are executed ('applied to the text').

fix-paragraphs.jsx
Paragraph breaks are indicated by double line breaks. Single paragraph breaks have no layout meaning in Latex, they can be used to make the source text more readable. The script therefore needs to delete single line breaks, though not of course those in environments such as tables, figures, lists, and math.

The script therefore marks up all the line breaks that should remain with a text condition. Then all line breaks that are unconditional – that is, those that are not in a text condition – are deleted.

tables.jsx
Table environments are converted to InDesign tables. They are placed as anchored floats on the pasteboard so that until you place them in the text they stay at their callouts.

Rules set in the Latex code are ignored, all tables are formatted with 0.75 pt top and tail rules and a 0.5 pt rule below the header. The script creates object, paragraph, cell, and table styles as well as a caption style.

If the caption command contains an item for the list of tables in the frontmatter, object and paragraph styles are created for it, too.

figures.jsx
Figures are placed as anchored floats on the pasteboard so that until you place them in the text they stay at their callouts. The figure file to be placed must be in a subdirectory `Images` in the active document's folder. If the graphic can't be found the figure is still placed (with caption, label, etc.) and graphic's name is placed in the figure. The script creates object and paragraph styles as well as a caption style.

If the figure's caption command contains a short title (an abbreviated caption for the list of figures in the frontmatter), a character style is applied that hides the short title and its brackets.

arrays.jsx
Arrays are placed as inline tables. A paragraph, cell, and table style are created by the script.

parbox.jsx

Paragraph boxes (`\parbox`) are placed as inline frames. The necessary styles are created on the fly. Width and vertical alignment are taken from the command's parameters, if present.

margin-notes.jsx
Margin notes are placed as anchored floats in the margin. Object and paragraph styles are created on the fly.

character-styles.jsx
Character formatting is handled by character styles. The following commands are converted:

```
\emph
\underline
\textit
\textup
\textbf
\textmd
\textsc
\textsf
\textsl
\texttt
```

```
\texttt
\textsuper
\textsub
```

For all these a character style is created with the command's name (without the backslash). The only formatting applied is underline so that you can see that some style was applied to some text. You'll have to apply the desired formatting after the conversion.

The commands `\sc`, `\it`, `\bf`, and `\it` are handled separately because they can appear in two formats: `{\it{a}}` and `{\it a}`.

A problem is that InDesign can't handle compound styles. That is, if you apply first italic then bold, you end up not with bold–italic, but with bold. So applying a character style to some text removes a character style that had been applied to that text earlier. Apart from super–sub, super–super (and all other variants of super- and subscript) in equations, compound character styles are not yet supported by the script.

A special case is `\text`, which is simply removed because in Latex it's the equivalent of character style [None].

Finally, if the document contains any double-struck letters (`\mathbb`) a character style is created (`mathbb`) and applied. You'll need to set a font in that character style if your base font doesn't contain double-struck letters.

symbols-accent-1.jsx

Accented characters can be entered in two different (though similar) formats. The first is used outside math environments and uses conventional symbols for accents (" for umlaut, u for breve, etc.).

Accented characters can be coded in several ways in the Latex file. I've come across three; for instance, the ô can be entered as `\^{o}`, `{\^o}`, and `\^o`.

Latex is quite flexible in that when a character isn't present in the active font, it uses the base letter and the floating ('combining') accent. InDesign isn't that flexible: when a character isn't present in the active font InDesign shows a pink triangle, but that's not so easy to detect in a script. The only way to determine whether a certain character is includ-

ed in the active font is a brutal test, which is not used here. Instead, the script enters a defined set of accented characters as such (characters from the Unicode ranges Latin-1 Supplement, Latin Extended-A, and Latin Extended-B), and for other combinations it places the base character and a combining accent. How well that works is entirely dependent on how well the combining accents were implemeted in the font. In my experience most modern OpenType fonts perform well in this respect.

symbols-accent-2.jsx
The second format uses full accent names (*grave*, *acute*) and must be used in math mode (not relevant in InDesign). Not all accents are available in this mode. The commands `\imath` and `\jmath` are replaced with the dotless i and j.

There is a large degree of overlap between the two notations and it's possible in principle to combine the two include files, but in order stay sane I keep them separate.

symbols-greek.jsx
A fairly straightforward conversion of `\alpha` etc. to α etc. Allowance is made for Greek letters with an argument (as in `\delta{E}`).

symbols-misc.jsx
A collection of symbols: digraphs (œ, æ), language-specific symbols (ł, å), symbols (dagger, various types of dash), and some occasional ones such as \og and \fg for the the guillemets (« and »).

Finally, hexadecimal codes are converted: `\symbol{“014B}` is replaced with Unicode character 014B (ŋ).

All these are simple replacements, and it's simple enough to add to the list in the script. The script code is quite straightforward.

symbols-ipa.jsx
This is a good example to show how easy the converter can be extended. Phonetic (IPA) characters can be handled in Latex packages (the included script uses the names from the TIPA package). To add support for other packages, write a separate script (or get it written) using

a format based on this include file and enter a call to the function in `latex-to-indesign.jsx`.

symbols-math.jsx
An incomplete collection of math symbols. This function handles quite a few but certainly not all. But it's easy to add more symbols, the code is pretty straightforward. The script handles simple math symbols and those with an argument (e.g. `\pm{12}`).

math-log-like.jsx
Letters and strings of letters are italicised in equations. However, mathematical designators (sometimes called log-like expressions) should not be italicised. The script applies strikethrough to these words so that they can be ignored by the math processors; the strikethrough is removed later. The words targeted here are `arccos`, `arcsin`, `arctan`, `arg`, `cos`, `sosh`, `cot`, `coth`, `csc`, `deg`, `det`, `dim`, `exp`, `gcd`, `hom`, `inf`, `ker`, `lg`, `lim`, `liminf`, `limsup`, `ln`, `log`, `max`, `min`, `Pr`, `sec`, `sin`, `sinh`, `sup`, `tan`, and `tanh`. You can add more if needed.

math-misc.jsx
Some miscellaneous math replacements that somehow didn't make it into any of the other math modules. The commands `\left` and `\right` are removed, they have no function for us in InDesign. Secondly, the character style text is created, enabling strikethrough, and is applied to all instances of `\text{...}`.

apply-math-character-style-to-missing-glyphs.jsx
The script applies the document's Math character style to all missing math glyphs in the UNicode ranges defined in the script.

math-processor-1.jsx
Some writers use math to apply italic outside math environments, for instance, `$a$` as an alternative to `\emph{a}`. Here we create a character style Italic and apply it to these fifth-column crypto math expressions. We do this to make life easier for the math processor that runs next.

In addition, a number of other math and math-like things are handled. For example, creative attempts at producing the degree symbol

(`^{\circ}`, `^\circ`) are taken care of, some different types of dash (en-dash versus the minus symbol), and `\pmod{...}` is changed to `(mod ...)`.

math-processor-2.jsx
The main math processor. The script creates these character styles:

```
italic
sub
super
subsub
supersub
supersuper
subsuper
sub italic
super italic
subsub italic
supersuper italic
subsuper italic
supersub italic
```

InDesign won't allow compound character styles, which means among other things that a subscript to a subscript needs a separate character style. The script sets the point size and the vertical offset for each of these styles, they can be adjusted later.

Various object, paragraph, cell, and table styles are created as well. They are needed for fractions, for instance, which are created as small single-column two-row tables. The cell styles used for these fractions set the dividing rule's appearance.

The script handles operator spacing, for which quart spacing is used (U+2005 in InDesign). Stretchy braces and parentheses are handled as well; the script uses the font STIXSizeTwoSym for these if it's present.

math-sqrt.jsx
Square roots are handled in a second pass (for better or for worse; we'll see). As elsewhere, object, paragraph, and character styles are created for these on the fly.

newtheorem.jsx
`\newtheorem` constructs are processed. Styles are created on the fly.

lists.jsx
`\itemize` and `\enumerate` environments are handled as expected. All styles are created on the fly.

lists-description.jsx
`\description` constructs are processed. Styles are created on the fly.

footnotes.jsx
Footnotes are rendered as InDesign footnotes.

footnotemarks.jsx
Footnote marks and footnote text, too, (distinct from `\footnote{}`) are rendered as footnotes.

heading-styles.jsx
The script creates paragraph styles for chapter, section, subsection, and subsubsection paragraphs and applies them. The styles are skeletal, they should be specified later.

index.jsx
Latex index markers are converted to InDesign topics and page references. You'll find them in the Index panel.

hyphenation-exceptions.jsx
The script looks for the file `preamble.txt` (in the active document's directory) and reads any hyphenation exceptions. The exceptions are applied in the text, replacing the (exception) hyphens with U+00AD, which is InDesign's discretionary hyphen.

last-act.jsx
Indents: the script applies the paragraph style `Body (no indent)` to paragraphs that contain the `\noindent` marker and Body paragraphs that follow a non-Body paragraph (a heading, a display equation, etc.).

Independent post-processing scripts

020-number-captions.jsx
The script numbers figure and table captions, using InDesign's automatic paragraph numbering. For each type (tables, figures) a separate list is created. If the document contains a chapter heading, the chapter number is used in thec captions (Section 3.1, Table 3.1, etc.).

021-caption-number-fix.jsx

An InDesign bug: on a page with two figures (or two tables) the numbering is the wrong way round (Figure 2.4 is followed by Figure 2.3 on the same page). The script fixes this by setting the numbers at each caption. The script needs to be rerun whenever figures and/or tables are added, moved, or removed. This script is needed only when you think that there can be two tables or two figures on the same page, but it does no harm to run it anyway.

030-cross-references.jsx
The script creates cross-references from Latex's `\ref{...}` and `\label{...}` commands and places them in anchored frames with a light-red shade on the pasteboard. The shade is removed from the frames of resolved references.

040-citations-and-bibliography.jsx
Using a Latex .bib file, the script resolves citations in the text and optionally generates a bibliography.

070-index-set-page-ranges.jsx
Latex uses a marker to indicate the end of a topic's range, while InDesign doesn't: a topic range is set by counting paragraphs or pages (and some other methods, which aren't relevant here). The convertor marked up the topics that range over some text, but cannot set the actual range until the document is finished.

This script should therefore be run immediately before the index is generated. It sets the page range at topics as a page count. Each time the document changes and the index is regenerated, the script should be run again.

Latex2InD.jsx
If the converter was unable to create an equation, this script can be used to run Latex in the background to create a PDF. See page 23 for details.

Revision history

6. 15 March 2025
- Added a chapter on how to use Latex2Ind (pp. 23–28) and added the script to the download.

5. 9 Feb. 2025
- Added a tip: the converter has no progress bar, but when you open GREP tab of the Find/Change window you'll see all the regular expression flash by that are used by the converter. Makes a good progress indicator because there are zillions of GREP searches and replacements.
- Added a section on InDesign pain points (p. 2).
- Added a note on line endings (p. 9).
- Added more elaborate paragraph numbering to section-heading styles and added a section outlining the numbering system (pp. 15–16).
- Improved the presentation of various lists; see p. 42 for details.
- Added support for the tabbing environment (p. 44).
- Some more document cleaning added.
- Bug fixes.

4. 22 Jan. 2025
- Added `\pageref` to cross-reference processing.
- Added support for various types of box (mbox, makebox, framebox, fbox).
- Expanded table processing: added support for column spans; included inline tables; improved the processing of column properties such as width and alignment. There is still some work to do on tables.
- The chapter on tables was substantially rewritten.

3. 6 Jan. 2025
- Bug fixes.
- Added support for column-spanning figures (figures containing more than one minipage environment).
- Added a script to apply a Math character style to all math symbols that can't be displayed using the default font.

2. 2 Jan. 2025
  – Bug fixes.
  – Added subscripts for log-like functions.
  – Sum, integral, and product are now sensitive to whether they are in a display or an inline equation; this determines the position of the arguments.
  – Added support for quotes and quotations.

1. 27 Dec. 2024 – first drop.