Novelang

The electronic document generator

Written by Laurent Caillette

Version 0.56.0

Introduction

Novelang transforms text files, written with a wiki-like syntax, into nicely-formatted documents like PDF or HTML, using customizable stylesheets. Novelang targets professional writers who need to focus on the content, letting the automated rendition do the boring job.

Novelang retains the spirit of compiler-based development tools, where the machine performs all possible validations before producing the final output. Novelang's unique fine-grained parsing grabs text details like parenthesized blocks or punctuation, and enforces consistent structure and rendering.

Novelang is a project started by Laurent Caillette, a Java architect who believes in the power of simple, reliable tools. You can mail him on Users mailing list or leave a comment on the Novelang blog (see links).

Novelang software is free to use and redistribute, under the terms of the GNU Public License v3.

Quickstart

Requirements: Java 6.

First, download the latest version of Novelang, and unzip it in some directory we'll call `$NOVELANG_HOME`.

First document

Just create a file with your favorite text editor and name it `hello.novella`. Content could look like this:

```
Hello, world!
```

Run!

Then, from the directory where the file is located, launch the Novelang dæmon:

```
java -jar $NOVELANG_HOME/lib/Novelang-bootstrap-$VERSION.jar httpdaemon
```

Open your favorite Web browser at this address: http://localhost:8080/hello.html

And you should see your text, with correct formatting! Even more: the same source text may render to a PDF document: http://localhost:8080/hello.pdf

Rationale

There are plenty of means to produce electronic documents and Novelang comes at the end of a long way across through various solutions.

WYSIWIG word processors

Their main problem is also their primary feature: they mix presentation with content. While this makes multi-format rendering horribly difficult, this also cripples the act of writing, making people focus on how the text appears instead of what it means.

WYSIWIG publishing tools

Those tools, like XPress and InDesign, are fine for high-end publishing with ultimate control of every aspect of how the document looks like. But they're definitely not content-oriented.

LaTeX

LaTeX got it right. It is based on plain text with simple markup and a powerful macro system. It exists from 20 years and is incredibly mature now.

Why not using LaTex? The markup is too verbose, it accepts macro language inside the content, and it's uneasy to setup. But if you have complex typography to automate (like for a book about mathematics) and skilled writers, it's definitely the right tool.

DocBook

DocBook is a semantic XML-based markup for technical documentation. It has many great ideas (Novelang borrows some of them) but fails to cover every case and is really too verbose. And it lacks an off-the-shelf solution.

Wiki

Wikis are a family Content Management Systems aiming at creating Web sites quickly. They rely on a simplified markup which can get incredibly productive. There is a lot to say about their syntax.

(See an overview of various wiki systems. See some interesting discussion about wiki syntaxes on Markdown and Wiki Creole and Coding Horror.)

Alas, wiki solutions don't care a lot about supporting accurate typography and most of them don't generate decent PDF.

Finally…

Novelang came after a pair of prototypes.

— One using Apache Forrest and DocBook.

— The other basing on Xilize, which transforms a Wiki-like syntax into XHTML, and Flying Saucer which generates PDF out of XHTML.

The Forrest-based solution supported document preview through a Web browser, which is incredibly productive. It also supported document fragment inclusion. The Textile-based one had a much more pleasant markup. So it seemed a good idea to start a project mixing the best of both worlds, with a tighter and cleaner syntax.

Frequently asked questions

Is there a chance to see a WYSIWYG editor for Novelang?

WYSIWYG (What You See Is What You Get) doesn't make much sense when output may occur in multiple formats. The closest thing that Novelang could offer is some plugins and configuration files for popular text editor supporting custom syntax highlighting.

Why so many versions?

Delivering by small increments makes bug detection and analysis easier. Novelang's automated release process makes easy and cheap to deliver a lot.

Is Novelang good enough for production?

Sure it is. The author is using it daily to dump the mess out of his mind.

How unit-tested is Novelang?

Each new version passes more than 700 non-regression tests.

Who's using Novelang?

Less than 3 people in the world.

Will Novelang become a commercial product?

The market is already full of content management systems and wiki stuff, so Novelang would have very low visibility. There are better chances to sell customization service. But monetization is not a priority.

Default PDF stylesheet is ugly. Any chance to fix this?

Yeah. This will occur one day.

XSL sucks. Is there another mean to define stylesheets?

Yeah it sucks but it offers a single, well-documented mean to support various rendition formats. Using XSL out-of-the-box helps to focus on other topics, like defining a correct syntax for source documents. In the future, Novelang may support a dedicated language, probably based on Clojure. There are already some interesting libraries for XML and HTML.

FO sucks. Is there any other mean to generate PDF?

FO is great for a certain kind of documents with simple and precise layout. FO handles hyphenation, page references to page numbers, various stylings, that are very complicated to make working when put together. So FO is here to stay. Please note that lot of FO ugliness comes from the XML syntax for writing XSL stylesheets.

What about supporting more character sets?

Because Novelang's grammar is very strict, it requires some changes to support new characters. Those will be added at user's request (already happened for Hungarian).

Release notes

Version number has the “Major, Minor, Fix” form. Before Novelang reaches version 1.0.0, a change in “Minor” means a new feature. A change in “Fix” means a bugfix, while retaining compatibiliy with documents and stylesheets.

Release notes do report when there is a possible compatibility break with existing documents and stylesheets.

0.56.0

— Added Romanian characters, as listed on Wikipedia. This includes diacritics made obsolete by the spelling reform of 1904, minus the d/D letter with a comma below, for which Unicode offers no precomposed characters.

0.55.0

— New `--temporary-dir` option. For better error messages, Novelang now buffers the whole document before sending it to the HTTP client. When the document is too big Novelang buffers it into a temporary file under this directory. If an error occurs then its not too late to send an HTTP redirection.

— Fixed SVG embedding for PDF. Now the image appears as true vector image inside the PDF. Correct reference to the SVG resource implied adding a `$content-directory` parameter passed to XSL stylesheets.

— Fixed loss of request parameters when issuing error page.

— Less verbose logging of Logback configuration at startup.

— Fixed various cases of bad problem reporting, where location in origin file was missing.

— Plenty of other small fixes.

0.54.0

— Because of some cleanup, `n:relative-identifier` becomes illegal in stylesheets (wasn't generated since a few versions). This might break a few existing stylesheets.

— Better logging of accessed resources.

— Documentation enhancements.

0.53.6

— Fixed resource loading bug appeared in 0.53.5, that caused to ignore overriden resources (like stylesheets).

0.53.5

— Various logging enhancements.
— Various enhancements to the documentation.
— Disabled full parsing and validation for SVG files.

0.53.4

— Fixed logging configuration with `--log-dir` option.

0.53.3

— Fixed barcode generation.
— Minor logging enhancements.

0.53.2

— Indicating error location when something goes bad during XSL transformation.
— Minor fixes on HTML documentation.

0.53.1

— Minor cosmetic changes for HTML generation.

0.53.0

— Experimental support for Multipage. See the result in Novelang documentation.
— Small logging enhancements.
— More restrictive rules when applying XSL stylesheets. Generation now breaks on warnings. This might break existing incorrect stylesheets.
— Changed default representation of Fragment Identifiers, both Implicit and Explicit. Removed leading double reverse solidus `\\` when rendering (still required in document sources).

0.52.0

— Added `n:block-inside-asterisk-pairs`. Default stylesheet render it as bold.

0.51.1

— Upgraded from FOP-0.95 to FOP-1.0. FOP is the library for generating PDF documents.

— Various other library upgrades that shouldn’t affect normal users.

0.51.0

— Fixed: list with double hyphen and number sign was using a “plus sign” everywhere (source documents and XML elements). This might break existing documents and stylesheet using this brand new feature.

0.50.2

— Fixed: support paragraphs as lists (`n:list-with-triple-hyphen` and `n:list-with-double-hyphen-and-number-sign`) inside `n:paragraphs-inside-angled-bracket-pairs`.

0.50.1

— Minor fix on `JavaShell` for cleaner shutdown when there is no default `JmxKit`. This only may affect users of Novelang-attirail subproject.

— Fixed documentation generation where release notes for SNAPSHOT versions appeared for non-SNAPSHOT versions.

0.50.0

— Embedded numbered lists (`n:embedded-list-with-number-sign`).

— Paragraphs as numbered lists (`n:list-with-double-hyphen-and-number-sign`).

— Switched to Maven 3. This required no change but future build features may not work with formerly-used Maven 2.2.1.

0.49.0

— In default stylesheet for HTML and PDF, the first `n:cell-row` element renders as a table header, if there is more than one. This might break existing documents.

0.48.0

— Fixed startup option in documentation.

— Tags and location for lines of literal. Required an intermediate `n:raw-lines` element nested inside `n:lines-of-literal`. This might break existing stylesheets.

— Location for cell rows with vertical line.

0.47.0

— Feature removal: relative identifier. Never used, and would make multipage rendering much more complicated.

— Small enhancements to Novelang-attirail, the reusable library.

0.46.1

— Added source packaging for Novelang-attirail subproject.

0.46.0

New experimental features for code reuse:

— Novelang-attirail subproject aggregating various tools. It’s not part of standard distribution, by now it requires separate rebuild.

— Pluggable logging implementation.

— Java code all under `org.novelang` package (was `novelang`).

0.45.0

— Added Greek and Polish characters to the grammar.

0.44.5

— Fixed release notes generation.

0.44.4

— Fixed a few references to old “Part” and “Book” terms, and file suffixes as well.

0.44.3

— Fixed Nhovestone report generation.

0.44.2

— Another fix for a build problem. Now the `deploy:deploy` goal should work properly when called from `release:perform`.

0.44.1

— Fixed build problem when deploying files and sending annoucements.

0.44.0

— Renamed Part into Novella and Book into Opus. Nicer, clearer. New recommended file suffixes are `.novella` and `.opus`. Old `.nlp` and `.nlb` suffixes still supported.

— Switched build system from Ant to Maven. Because all jars stand in the `lib` directory, this impacts the command line for launching Novelang.

0.43.0

— Added `nohead` option to `insert` command.
— Fixed some bugs around identifiers.
— Introduced detection of colliding explicit identifiers. This has no useful purpose for now but will serve as a basis for implementing internal links.
— Small performance enhancement on HTML document rendering in a Web browser: don’t use JavaScript to set collapsible descriptors hidden.

0.42.0

— Now requires Java 6.
— New Nhovestone report: Novelang has its own benchmark!
— Added stylesheet `html-FR.xsl` for French punctuation.
— Performance enhancement on rendered HTML page: when containing many tags it should load faster. Instead of dynamically computing styles on the Web browser, HTML rendered by the server directly includes those styles.
— Various performance enhancements on document generation. With the same amount of memory (`-Xmx` parameter), Novelang handles documents twice bigger and serves them 20 % faster than previous version. Benchmark ran against version 0.41.0 and 0.38.1. This includes buffered reading of Part files, multithreaded Part rendering, and reduced memory consumption when dealing with AST (Abstract Syntax Tree).

0.41.1

— Fixed bug with Promoted Tags, not detected under some circumstances.

0.41.0

— New feature: Promoted Tags. Implicit Tags matching Explicit Tags become Promoted Tags.
— Support lines of literal inside paragraphs inside angled bracket pairs.
— Minor enhancements on HTML default stylesheet.

0.40.1

— Fixed display bug on generated documentation.

0.40.0

— Brand new stylesheet for HTML.

0.39.2

— Reject diacritics in tags.

— Filter on implicit tags as on explicit tags.

0.39.1

— Fixed documentation generation.

0.39.0

— Experimental feature: descriptors in default HTML stylesheet.
— Support Unicode files starting with a BOM (Byte Order Mark).
— Reject TAB character.
— Display Unicode name of a rejected character (as long as it belongs to the set of 16 bit Unicode characters as defined in Unicode 5.2 standard). This feature is experimental and may wreck existing error messages.

0.38.1

— Fixed occasional crash caused by Implicit Tags.

0.38.0

— Experimental support for Implicit Tags, deduced from level's title.

0.37.0

— Experimental support for Implicit Identifiers, deduced from level's title. See “Identifiers” chapter in Part syntax.

0.36.0

— New `--style-dirs` command line parameter (superceding `--style-dir`) for multiple style directories.
— Minor identifier-related fixes and internal refactorings.

0.35.0

— Experimental support for identifiers. See “Identifiers” chapter in Part syntax, and “Insert Command” in “Book Files”.

0.34.1

— Fixed: bug preventing from starting a Novelang release with a numbered version.

0.34.0

— New `levelabove` option for `insert` book command.

— New `sort` option for `insert` book command.

— New `explodelevel` batch command for splitting one document's levels into several parts.

— New `--content-root` command line argument for setting the directory where content files reside.

— Fixed: paragraph as list did not support indented embedded list items.

0.33.1

— Enhanced error reporting when parsing an ill-formed Book file.

— Added automatic whitespace between punctuation sign or various blocks, and leading apostrophe.

0.33.0

This version introduces changes that may break existing documents and startup scripts.

— New document extension `.fo`, for debugging FO stylesheets.

— Removed the dollar sign `$` from Book syntax.

— Removed `section` command from Book syntax. No remplacement planned in the short term.

— Removed `title` command from Book syntax. Use document parametrization instead (see how Novelang documentation works).

— Renamed `--serve-localhost-only` to `--serve-remotes` and removed the boolean argument.

— Fixed some bugs. Books now scan a directory correctly when `recurse` option is not set.

0.32.1

— Fixed `--serve-localhost-only` command-line option: now works with IPv6.

0.32.0

— New `--serve-localhost-only` command-line option for HTTP dæmon.

— Minor fixes around block after tilde. They may appear in blocks inside hyphen pairs.

— Minor enhancements to build script: Automated blog posting for a new release. Display version number in documentation.

0.31.1

— Minor fixes around block after tilde. They may appear inside embedded lists and block inside solidus pairs.

— Better handling of automatic space addition.

— Now character table contains decimal values.

0.31.0

This version introduces changes that may break the rendering of existing documents.

— New feature: Block after tilde. This supercedes proximity rules for blocks inside grave accents and blocks inside grave accent pairs.

— Fixed some minor bugs occuring under Windows®.

0.30.0

— Bug fixes: Handle more URL: HTTPS, non-standard query parameters. Fixed disappearing whitespace between word and block inside grave accents, when inside level title or embedded list.

— Minor enhancements: Better sizing of embedded lists with default CSS. Documented some Windows tips. Added registration sign and copyright sign as supported characters in source documents.

0.29.0

This version introduces changes that may break existing documents.

— Better handling of spaces inside blocks of literal (trimming, collapsing, replacing by no-break spaces).

— Consecutive blocks of literal get a zero-width space automatically inserted inbetween.

— When a block of literal inside grave accents is immediately preceded/followed by a word, a zero-width space gets automatically inserted inbetween.

— In case of a word starting/ending with an apostrophe, the rendering adds a whitespace to separate from the preceding/following word.

— Added the color palette editor

0.28.0

— Tag-based document filtering supported in URL parameters (`tags=tag-1;tag-2`). This is faster than JavaScript-based one, and makes the feature available to PDF and other formats.

— Report line number in case of unclosed block delimiter.

— Better logging: more concise messages, added some missing stuff.

0.27.1

— Fixed MIME type of documents served by HTTP dæmon.

0.27.0

— Experimental support for tag-based document filtering with default HTML stylesheet. This is purely Javascript-based (runs all in the browser) and performance are disastrous on big documents.

— Fixed minor parsing bugs around URL.

0.26.0

This version introduces changes that may break existing stylesheets.

— Introducing tags: content annotations.

— Fixed some grammar bugs with URL, embedded lists and various line break configurations inside non-symmetrical delimiters (pairs of solidus, double quotes, hyphen pairs).

— Default stylesheets moved to top level (no more `style/default` directory). The `default` word becomes the prefix (it was formerly the suffix) so `html-default.xsl` is now `default-html.xsl`.

0.25.0

This version introduces changes that may break existing stylesheets.

— Simpler URL naming: no longer requires a line break preceding the block containing URL name.

— As a consequence, `n:external-link` becomes `n:url`; `n:url` becomes `n:url-literal`; `n:link-name` becomes `n:block-inside-double-quotes` or `n:block-inside-square-brackets`.

0.24.0

— Experimental support for embedded lists: hierarchical lists inside paragraphs.

0.23.0

— Named URL.

— `n:url` element gets wrapped into `n:external-link`.

0.22.0

This version introduces changes that may break existing documents and stylesheets.

— Fixed: Book now display images correctly.

— Fixed: Safari 4 did not render SVG resources when referenced from HTML document.

— Fixed: HTML document now aware of SVG image dimension.

— Slight change: `n:pixel-width` and `n:pixel-height` become `n:image-width` and `n:image-height`, with measure unit if available. For raster images, it is always `px` unit.

0.21.0

— Experimental support for raster and vector images. Known limitation: only works for Parts.

— Fixed argument parsing of batch argument generator (reported by Lmre).

0.20.0

— Tables! See part syntax for details.

— Documentation now lists all characters allowed in a source document (see at the end).

— Default stylesheet are now modular: all logic is contained by bundled `[pdf,html,nlp]-default.xsl` while the stylesheet taken where no stylesheet name is specified remains `[pdf,html,nlp].xsl`. This makes possible to override defaults without specifying a new stylesheet name.

0.19.0

— Default charset for source documents is now UTF-8.

— Every character may be escaped using its Unicode name.

— Batch document generator.

— HTTP dæmon now started with additional `httpdaemon` command name.

The new command-line syntax now is:

```
java -jar ...novelang-VERSION.jar httpdaemon <options>
java -jar ...novelang-VERSION.jar generate <options>
```

0.18.0

— Experimental support for custom charset for both source and rendered documents.

0.17.0

This version introduces changes that may break existing documents and stylesheets.

— Renamed `n:level-description` to `n:level-title`.

— Unlimited level nesting.

0.16.0

This version introduces changes that may break existing documents and stylesheets.

— Brand new naming scheme for syntactic nodes!

— Chapters and sections don’t exist anymore. All what stylesheets will see are “levels”.

— Top-level delimiter (formerly “chapter”) in Part now starting with two equal signs == instead of three asterisks.

— Character escape now basing on Unicode name, plus optional HTML entity name.

— Option `createchapter` of the `insert` command renamed to `createlevel`.

0.15.0

— Stylesheets containing XPath expressions relative to non-existing grammar token are detected and rejected.

— Added new stylesheet: `html-source.xsl`. It generates HTML source in HTML with special markup highlighted. This is especially useful for posting raw HTML on Blogger while editing the text with Novelang!

— ANTLR grammar refactoring continues, now tokens and supported characters are described in Java code generated from the grammar.

— Fixed: Font list regression, now displays all characters as in Novelang-0.13.0.

— Fixed: now some broken Part causes the whole Book document generation to fail when it's a recursive include.

0.14.0

— Refactored ANTLR grammar for supporting some planned features. Known regression: some useful characters missing from font list.

0.13.0

— Better detection of text inconsistencies (lexer now reports problems).

— Enhanced font list: doesn't break where there are no custom font at all.

— Cleaned up `samples` directory: most of stuff here was for tests, that moved to `src/test-resources`.

0.12.0

— Enhanced font list: looks better, displays more useful characters, reports broken fonts.

— The `timestamp` parameter passed to stylesheet is no longer of `java.lang.String` type, now it is a `org.joda.time.DateTime` for more formatting options.

— Option `createchapters` of the `insert` function renamed to `createchapter` as it is no longer limited to recursive addition; now it applies to single-part insert.

0.11.0

— Multiple font directories. No longer need to give special names to font files (internal name recognized automatically).

— Normalized command-line arguments, no more system properties.

— No longer need for font-metric files.

0.10.0

— Barcode generation in PDF with Barcode4J.

— Fixed character escaping with HTML.

0.9.0

— Custom fonts for PDFs.
— Hyphenation for PDFs.
— New `$style` parameter for the `insert` function.
— Superscript.
— XSLT extension: plain text numbering. See chapter on custom stylesheets.
— Fixed import bug when using `punctuation-FR.xsl`.
— Documentation updates.

0.8.0

— Directory listing.
— New `--port` option for `HttpDaemon`. Useful when default TCP port (8080) is busy.
— Fixed mispelling of “literal” (was “litteral”).
— Plenty of bug fixes.

0.7.0

— Enhanced literal and character escaping.

0.6.0

— Multiple stylesheets.

0.5.0

— First public release.

Architecture

Novelang runs as a batch tool, or as an HTTP dæmon. Both use the same engine, passing URL-like requests (like `/doc/Novelang.pdf`).

The rendition of a document occurs through 3 mains stages:

- Source parsing.
- Tree mangling.
- Rendering.

Source parsing

Source parsing is about reading a source files into an AST (Abstract Syntax Tree). The AST is made of nodes with a text payload, an optional type label, and zero or more children.

The parser relies on ANTLR-generated code.

Source parsing occurs in parallel when a single Opus file includes several Novella files.

Tree mangling

The tree mangling reorganizes the AST before rendering. This occurs in many steps.
Some of those steps are:

- Integrating double-quoted text as `n:url-literal` inside `n:url` nodes.
- Building hierarchies for levels and lists.
- Finding identifiers and detecting identifier collisions.

Rendering

Rendering converts the AST to a human-readable format, like PDF or HTML. There is a built-in renderer relying on XSL but as the Renderer is basically a function taking an output stream and an AST as input parameters, rendering can occur in virtually any format.
split the document into multiple pages (through a customizable stylesheet).

Statelessness

Novelang applies many features of functional programming. Most of Novelang’s internal components don’t retain any state past initialization. Most of data structures are immutable, especially the AST. (This required to develop specific algorithm to deal with immutable trees.)
Because Novelang’s code avoids unnecessary side-effects, this greatly improves its inherent stability.

Directory layout

Novelang searches all its files in various directories. There are options to customize those directories, but it’s convenient to know about the defaults.
Typical layout of a Novelang project looks like this:

```
/
  opus.opus                   <- Opus file
  style/                      <- Directory containing styles
    pdf.xsl                   <- A stylesheet file
    html.xsl
  fonts/                      <- Fonts directory
    Garamond.ttf
    Garamond.bold.ttf
    Garamond-bold-italic.ttf  <- a font file
    Garamond-italic.ttf
  developer-manual/
    chapter-1.novella         <- A Novella file
    chapter-2.novella
    chapter-3.novella
  hyphenation/                <- Hyphenation directory
    en_US.xml                 <- Hyphenation definition
    hyphenation.dtd           <- Mandatory file
  user-manual/
    chapter-1.novella
    chapter-2.novella
```

General syntax

Novelang recognizes Novella files with the `.novella` suffix. Novella files are plain text files, containing pure textual content, plus a limited amount of decorations to help Novelang to structure the text prior to rendering.

Here is a valid Novella file. It should look familiar to people who know wiki syntax:

```
== Title of level 1

This is a first paragraph on
two lines.

This is a `block of literal#@&)` inside a paragraph.
//This other block will show in italics//.


=== Title of level 2

<<
This is a quoted paragraph.

This is a second quoted paragraph.
>>
```

See? Decorations look “imaged” like ASCII emoticons. Equal signs figure indents for the title level, and angled brackets look like opening/closing quotes. Unlike with HTML or LaTeX, it’s easy to read a Novelang source document.

Novelang makes a great effort for making its grammar consistent. Before a detailed presentation of all available decorations, here are the fundamental notions to deal with.

— Paragraphs. The central notion of Novelang's Novella grammar is the paragraph. A paragraph is a sequence of textual items kept together because there is *no more than one line break at a time*. So a paragraph cannot contain two consecutive line breaks, or it would be split in two paragraphs. Paragraphs mainly contain words, punctuation signs, blocks, list items and external links.

— Blocks. A block is a subset of a paragraph (for things like text in parenthesis). Blocks may contain blocks. Because blocks occur only inside paragraphs, they cannot contain two consecutive line breaks.

— Literal. Literal is text with uninterpreted characters. Novelang supports several kinds of literal, whether it is inside a paragraph or outside.

— Levels. A level is a hierarchical container, that carries meaning about text structure. Levels represent things like chapters and sections.

Detailed syntax

Now let's discover all decorations supported by a Novella.

Along with their syntax is given the XML element names usable in a XSL stylesheet. Names may seem weird at the first glance. A block inside two pairs of solidus `//` is called `n:block-inside-solidus-pairs`. Why not calling it simply "italics"? First, keep in mind that those XML names only appear inside custom stylesheets so you may not care about them at all. Novelang transforms the source document in an abstract tree before rendering it through a stylesheet. If you create your own stylesheet you can process a block inside `//` like a footnote in super-bold or whatever, then "italics" would be quite confusing. So XML element names don't try to carry assumptions about the usage of the element. It just describes the originating decoration.

Paragraph, regular

XML element: `n:paragraph-regular`

A regular paragraph is made of contiguous lines of text (two consecutive line breaks cannot occur inside a paragraph). A paragraph may contain words, punctuation signs, external links, list items and blocks.

With Novelang default stylesheet, such text is rendered as normal text:

```
First
paragraph.

Second paragraph.
```

Levels

XML elements: `n:level`, `n:level-title`

Levels are delimited with a simple syntax, using a separator = telling about the depth of the level. A Novella contains up to three levels, including level 0 which is the default.

Considering a Novella like this:

```
Text at depth 0.

== Depth 1

Introductory text.

=== Depth 2

Blah blah blah.

=== Depth 2

Blah.

== Depth 1 again

=== Depth 2

...
```

The level structure of the Novella looks like this:

```
+ n:novella
  + n:paragraph-regular "Text at depth 0"
  + n:level
  | + n:level-title "Depth 1"
  | + n:paragraph-regular "Introductory text."
  | + n:level
  | | + n:level-title "Depth 2"
  | | + n:paragraph-regular "Blah blah blah."
  | + n:level
  |   + n:level-title "Depth 2"
  |   + n:paragraph-regular "Blah."
  + n:level
    + n:level-title "Depth 1 again"
    + n:level
      + n:level-title "Depth 2"
      + n:paragraph-regular "..."
```

As explained later in this document, the depth of levels may be changed (increased) at Opus level. This is useful for creating documents which have great depth, while keeping edited content with at reasonable depth.

Inside a Novella, it’s incorrect to declare a first level with a greater depth than following one. The case below will cause an error:

```
== Depth 2, incorrect

=== Depth 1
```

Block inside solidus pairs

XML element: `n:block-inside-solidus-pairs`

Two pairs of solidus `//` may enclose a block of text.

With Novelang default stylesheet, such text is rendered as italics:

```
There are //italics//.
```

Block inside asterisk pairs

XML element: `n:block-inside-asterisk-pairs`

Two pairs of asterisk `**` may enclose a block of text.

With Novelang default stylesheet, such text is rendered as bold:

```
This is **bold**.
```

Block inside double quotes

XML element: `n:block-inside-double-quotes`

Two double quotes `"` may enclose a block of text.

With Novelang default stylesheet, such text is rendered inside double quotes (the character may vary depending on the language):

```
There are "double quotes".
```

Block inside square brackets

XML element: `n:block-inside-square-brackets`

Two pairs of square brackets `[` and `]` may enclose a block of text.

With Novelang default stylesheet, such text is rendered inside square brackets:

```
There are [square brackets].
```

Block inside parenthesis

XML element: `n:block-inside-parenthesis`

Two pairs of square brackets `(` and `)` may enclose a block of text.

With Novelang default stylesheet, such text is rendered inside parenthesis:

```
There are (parenthesis).
```

Block inside hyphen pairs

XML elements: `n:block-inside-hyphen-pairs`, `n:block-inside-two-hyphens-then-hyphen-low-line`

Two pairs of hyphen minus `--` may enclose a block of text.

There is an alternative where the ending delimiter is an hyphen then a low line `_`.

With Novelang default stylesheet, such text is rendered inside a pair of en dash characters (or em dash depending on language). This creates an interpolated clause. With the ending low line, there is no visible ending.

```
See -- interpolated clause -- here.
See -- no visible end -_.
```

Block of literal inside grave accents

XML element: `n:block-of-literal-inside-grave-accents`

Two grave accents ` may enclose a block of text containing characters which are not allowed otherwise (because they serve other Novelang grammar's purpose).

With Novelang default stylesheet, such text is rendered as normal text:

```
Almost `4ny- ch@rac7er 0.0.0 "'&#^> / *`
```

Spaces are handled in a special manner:

- Leading and trailing spaces are trimmed.
- Consecutive spaces between two characters are collapsed into one single space.
- Spaces are replaced by no-break spaces.

With the low line character _ figuring a no-break space, here is a sample of space replacement:

```
`  foo   bar ` becomes `foo_bar`
```

Block of literal inside grave accent pairs

XML element: `n:block-of-literal-inside-grave-accent-pairs`

Two pairs of grave accents `` may enclose a block of text containing characters which are not allowed otherwise (because they serve other Novelang grammar's purpose).

With Novelang default stylesheet, such text is rendered as monospaced text:

```
Almost ``4ny- ch@rac7er 0.0.0 "'&#^> / *``
```

Spaces are handled the same way as with block of literal inside grave accents.

Block after tilde

XML elements: `n:block-after-tilde`, `n:subblock`

A tilde character ~ may prefix a block of text containing no space nor line break. Inside the same block there can be other tilde characters.

Considering this source document:

```
~one(single)~block!
```

The internal structure looks like this:

```
+ n:block-after-tilde
  + n:subblock
  | + "one"
  | + n:block-inside-parenthesis
  |   + "single"
  + n:subblock
    + "block"
    + n:punctuation-sign "!"
```

Subblock may contain:

- Plain words.
- Punctuation signs.
- Block inside grave accents.
- Block inside grave accent pairs.
- Block inside parenthesis.
- Block inside solidus pairs.

When inside block inside solidus pairs, a whitespace *must* follow the block after tilde:

```
//~(un)ambiguous //
```

With Novelang default stylesheet, the subblocks are separated with a zero-width space. This is fine for overriding whitespace addition for typographic effect(s).

Paragraphs inside angled bracket pairs

XML element: `n:paragraphs-inside-angled-bracket-pairs`

Two lower-than signs << and two greater-than signs >> may enclose one paragraph or more. The angled bracket delimiting the paragraphs take place on the first column of the line, and there must be no other character on the same line.

Whith Novelang default stylesheet, such text is rendered as quoted paragraphs:

```
<<
First paragraph.

Second paragraph.
>>
```

List with triple hyphen

XML elements: `n:list-with-triple-hyphen`, `n:paragraph-as-list-item`

A triple hyphen `---` may start a paragraph, which becomes a list item. The triple hyphen must take place on the first column of the line. The paragraph as a list item follows the same rules as a regular paragraph.

With Novelang default stylesheet, such a paragraph is rendered as a list item, preceded by an em dash character:

```
--- First list item.

--- Second list item.
```

As XML element, the paragraph itself is a `n:paragraph-as-list-item`, while the sequence of items gets wrapped into a `n:list-with-triple-hyphen` element.

List with double hyphen and number sign

XML elements: `n:list-with-double-hyphen-and-number-sign`, `n:paragraph-as-list-item`

A double hyphen and a plus sign `--#` may start a paragraph, which becomes a list item. They must take place on the first column of the line. The paragraph as a list item follows the same rules as a regular paragraph.

With Novelang default stylesheet, such a paragraph is rendered as a numbered list item:

```
--# First list item.

--# Second list item.
```

As XML element, the paragraph itself is a `n:paragraph-as-list-item`, while the sequence of items gets wrapped into a `n:list-with-double-hyphen-and-number-sign` element.

Embedded list

XML elements: `n:embedded-list-with-hyphen`, `n:embedded-list-with-number-sign`, `n:embedded-list-item`

Inside a paragraph, a single hyphen `-` or a number sign `#` declares a list item. A list item is made of one single line (a line break would be interpreted as the end of item). An embedded list may declare subitems, with a greater indentation than containing item. All items of the same list have the same leading character (the behavior when mixing them remains undefined at this time).

With Novelang default stylesheet, the embedded list is rendered as a list, supporting nested elements (up to 3 levels with PDF, unlimited levels with HTML):

```
One embedded list:
- Item one.
- Item two.
  - Item two one.
  - Item two two.
- Item three.
And another one, in the same paragraph
# Item one.
# Item two.
  # Item two one.
End of paragraph.
```

As XML elements, the list is wrapped in a `n:embedded-list-with-hyphen` or `n:embedded-list-with-number-sign`, while each item gets wrapped into a `n:embedded-list-item` element. Sublists are wrapped the same way.

Lines of literal

XML element: `n:lines-of-literal`, `n:raw-lines`

Three lower-than signs <<< and three greater-than signs >>> may enclose one line of literal text or more. The angled brackets delimiting the paragraphs take place on the first column of the line, and there must be no other character on the same line.

The literal text may contain any character, including escaped characters and greater-than signs, as long as they don't form an ending delimiter (if you need to display that, then use an escaped character).

With Novelang default stylesheet, such text will be rendered verbatim, with a fixed-width font:

```
<<<
Here is literal.
  Indentation will be kept.
>>>
```

The `n:raw-lines` element appears wrapped inside the `n:lines-of-literal` element. Such an enclosed element is useful as a placeholder for tags.

Word after circumflex accent

XML element: `n:word-after-circumflex-accent`

A circumflex accent `^` may introduce a word immediately following another word.

With Novelang default stylesheet, the second word is is displayed as superscript:

```
April, the 1^st.
```

URL

XML elements: `n:url`, `n:url-literal`

A URL is primarily made of URL literal, which starts at the first column of a line, with nothing else on the same line (except trailing whitespaces).

When the URL literal is preceded by a block inside double quotes, or a block inside square brackets, it becomes a child of the `n:url` element.

With Novelang default stylesheet, the external link will show as a hyperlink named "here":

```
So you can click "here"
http://novelang.sourceforge.net
.
```

If the block inside double quotes, or the block inside square brackets must appear verbatim, then break the relationship by inserting some text element which won't affect rendering.

```
No "name" ` `
http://novelang.sourceforge.net
```

Tables

XML elements: `n:cell-rows-with-vertical-line`, `n:cell-row`, `n:cell`

A pair of vertical lines delimits a “cell”, which contains words, punctuation signs and various blocks, but no line break. The first vertical line must appear on the first column.

Several cells can be chained on the same line, using one additional vertical bar each time, thus forming a cell row.

One cell row, or a sequence of cell rows separated by line breaks, appear wrapped in a `n:cell-rows-with-vertical-line` element.

With Novelang default stylesheet, such a sequence of cell rows is arranged as a table:

```
| row1, col1 | row1, col2  | row1, col3 |
| row2, col1 |  row2, col2 | row2, col3 |
```

Tables behave like paragraphs in the sense they must be separated of other paragraph-like stuff by a pair of line breaks.

Raster images

XML elements: `n:raster-image`, `n:resource-location`, `n:image-width`, `n:image-height`

Raster images are also known as “bitmap” images, which are made of a grid of pixels.

The file path of an image represents the image to appear in the document. It may only appear as a paragraph (two line breaks separating from other paragraphs), or inside a table cell.

The image must be in JPEG, PNG or GIF format, with the extension being one of `.jpg`, `.png`, `.gif` respectively.

```
./orchid.jpg

./flowers/tulip.png

../animals/guinea-pig.jpg

| /images/logo.gif |
```

The path must start with zero, one, or two full stops, immediately followed by a solidus. Then follow optional directories, and the file name itself with its extension.

- If the image file is in the same directory as source document, then its name must start with a full stop and a solidus `./`
- Parent directories are referenced through double full stops then solidus `../`
- The project directory is considered as the root directory. Attempting to reference a directory above the project directory will produce an error. A path relative to the project directory (instead of being relative to the source document) starts with a solidus.

With Novelang default stylesheet, the image appears inside the document.

Under the `n:raster-image` element, the `n:resource-location` gives the image path relative to the project root. The `n:image-width` and `n:image-height` elements give the with and the height of the image, respectively, in pixel units.

Vector images

XML elements: `n:raster-image`, `n:resource-location`

Vector images work the same way as raster images. The image must be in SVG 1. 1 format, with `.svg` as extension.

```
./stars.svg
```

The with or the height of the vector image is copied from the SVG file, including units. Coordinates and lengths in SVG are explained here.

SVG documents may reference an external entity with a public identifier like `-//W3C// DTD SVG 1.1"` or `-//W3C//ENTITIES SVG 1.1` or `-//W3C//ELEMENTS SVG 1.1`. Other external entities are not supported yet.

Tags

XML elements: `n:explicit-tag`, `n:promoted-tag`, `n:implicit-tag`

A tag is a textual marker attached to some piece of a source document. It is made of a commercial at `@`, immediately followed by letters, digts, and hyphen minus (hyphen minus may only appear between letters and digits and there cannot be two consecutive ones). The tag appears immediately before tagged content (no more than one line break separating them).

By now, following Novelang constructs support tags:

- Level.
- Paragraph.
- Paragraph as list item.
- Paragraphs inside angled bracket pairs.
- Cell rows with vertical line.

A tag is not meant to be content by itself, but it may help to categorize the content, or provide additional information to the stylesheet.

Tags are passed as parameters to the HTTP dæmon or the batch generator, with the `tags` argument name, and with a semicolon as delimiter:

```
my-document.html?tags=TAG-1;TAG-2
```

With Novelang default stylesheet for HTML, the tags are rendered as tiny colorful floating rectangles in the right margin. A list of available tags appears in the topright corner in a disclosure blox. Checking / unchecking tags causes the document to display only tagged text, by requesting a document with the new URL.

Default stylesheet for PDF doesn’t render tags in a particular manner. But, as expected, if some text is excluded because it has none of requested tags, it will be excluded from rendition.

Tag usage sample:

```
  @my-tag @Level
== Tagged level

  @my-tag @some-other-tag
My tagged paragraph.

== Level: no explicit tag
```

The XML structure of source document above looks like this:

```
+ n:novella
  + n:level
    + n:explicit-tag "my-tag"
    + n:explicit-tag "Level"
    + n:level-title "Tagged level"
    + n:regular-paragraph
      + n:-explicit-tag "my-tag"
      + n:explicit-tag "some-other-tag"
      + "My tagged paragraph."
  + n:level
    + n:promoted-tag "Level"
    + n:implicit-tag "noExplicitTag"
    + n:level-title "Level: no explicit tag"
```

Level titles may also convert to Implicit Tags or Promoted Tags. Implicit Tags don't appear in the Tag list, but filtering on a given Tag retains document fragments tagged with Implicit Tags.

Implicit Tags come out from level title from a few simple transformation rules.

- Convert every letter to its form without diacritics.
- Whitespaces disappear.
- Punctuation signs and delimiters disappear. They become Tag boundaries.
- The first letter of a word that was preceded by another words gets uppercased.

Promoted Tags are Implicit Tags that match some Explicit Tag defined elsewhere in the document.

Here are a few transformation samples for Implicit (and Promoted) Tags:

Original Novelang source	Implicit Tags
This is some text. Be cool.	`@BeCool` `@ThisIsSomeText`
This is a title... (So what?)	`@ThisIsATitle` `@SoWhat`
version \`0.1.2.3\`	`@version0-1-2-3`
Some \`\`@#!garbage)<--.§\`\`here!	`@Some-garbage-Here`

Identifiers

XML element: `n:implicit-identifier`, `n:explicit-identifier`, `n:colliding-explicit-identifier`

An identifier is a textual marker identifying a level inside a Novella. The `insert` command (explained in “Opus syntax” chapter) recognizes identifiers to insert only given levels. An identifier starts with a double reverse solidus `\\`. The rest of the identifier is made of letters, digts, and hyphen minus (hyphen minus may only appear between letters and digits and there cannot be two consecutive ones).

Because often, titles are good candidates for identifiers, they resolve implicitely as identifiers as long as they are unique. As a result of Novella processing, an identifier may be either explicit or implicit. All implicit identifiers resolve as absolute ones.

Given following Novella file:

```
Paragraph 0

  \\One
== Level 1


  \\One-one
=== Level 1.1

Paragraph 1.1


=== Other

Other level (2)


=== Level 1.2

Paragraph 1.2

==== Level 1.2.1

Paragraph 1.2.1

==== Other

Other level (2)
```

It has:

- 2 explicit identifiers:
  - `\\One`
  - `\\One-onz`
- 3 implicit identifers:
  - `\\Level-1_2`
  - `\\Level-1_2_1`

Implicit identifiers do exist as long as they don’t collide with another identifier. That’s why `Other` doesn’t appear as an implicit identifier.

Novelang checks uniqueness of explicit identifiers within the same Novella. Since Novellæ are created independantly, identifiers may collide when aggregating Novellæ into a single Opus. With Novelang default HTML stylsheet, colliding identifiers show striked out.

Escaped characters

For displaying character that have a special meaning in the rendered documents, they must be escaped. An escaped character is enclosed into a left-pointing double angle quotation mark « and a right-pointing double angled quotation mark ». The escape code is the unicode name. HTML entity name is supported as well.

All characters that may appear in a source document are listed in an appendix, along with their escape codes.

Using those characters is exceptional, however – except in Novelang documentation! – as most of useful character are accessible as literal.

Comments

Sometimes it is useful to tell Novelang to ignore some lines in a Novella files.

Line comments begin with double percent sign `%%`.

Blocks comments are delimited by a pair of double accolades `{{` and `}}`.

```
%% This line is commented.

{{ These two lines are
commented. }}
```

Opus files

Opus files aggregate novella files. Opus files have the `.opus` suffix (like NoveLang Opus).

Command: `insert`

Opus files are useful when there is too much text to fit inside one Novella file. It is also easier to reorganize small Novellas by changing their order in a Opus file than copy-pasting inside one big file.

When Novella files become too numerous, Opus files can refer to multiple Novella files at once. Referring to project layout, here is a well-formed Opus file:

```
insert file:path/to/myfirstnovella.novella

insert file:path/to/mysecondnovella.novella
```

When the Novellas are located in the same directory as the Opus file, the single dot notation (for current directory) is supported:

```
insert file:.
```

The `recurse` option scans novellas in subdirectories.

```
insert file:. recurse
```

The `createlevel` option adds the content of the file under one new level. The level title is the filename, path and `.novella` extension being trimmed.

The `nohead` option takes effect when using identifiers (explained later). When the identifier references a level, this options causes the level title to disappear. The `nohead` option is mutually exclusive with the `createlevel` option.

The `sort` option provides a sorting method when inserting several files.

```
insert file:. sort=path+
```

The `sort` option determines the ordering of Novellas when there are many. Currently supported values are:

- `path+` for ascending sort on path name.
- `path-` for descending sort on file name.
- `version+` for ascending sort on version number.
- `version-` for descending sort on version number.

Version number if for files names of `major.minor.fix.novella` format, where `major`, `minor` and `fix` are positive numbers (this is Novelang's format for version numbers by the way).

The `levelabove` option adds the content of the file(s) under the last previously created level.

```
insert file:some-novella.novella levelabove=2
```

The `style=...` assignment adds a `STYLE` node to each of the trees corresponding to added novella.

```
insert file:.
  style=mystyle
```

Expert users will find this is useful for customizing the output right from an XSLT stylesheet:

```
<!-- Process styled chapter: -->
<xsl:apply-templates select="//n:chapter[n:style='mystyle']" />

<!-- Don't process styled chapter: -->
<xsl:apply-templates select="//n:chapter[not(n:style!='')]" />
```

A sequence of composite identifiers (as defined in “Novella syntax” chapter) limits the insertion of reference Novella to some fragments.

```
insert file:my-novella \\Some-identifier \\Some-other
```

Command: `mapstylesheet`

The “mapstylesheet” command defines a stylesheet for one or more MIME type. Referring to project layout above, here is the command to insert in `opus.opus` file in order to render HTML documents with `html-beautiful.xsl` and PDF documents with `pdf-beautiful.xsl`.

```
mapstylesheets
    html=html-beautiful.xsl
    pdf=pdf-beautiful.xsl
```

Internationalization

Novelang aims to support a wide range of languages, at least those with Roman characters. There are three aspects to consider:

- The character set of the source document(s).
- The character set of the rendered document.
- The characters recognized by Novelang grammar.

Taking advantage of the underlying Java platform, Novelang supports numerous charsets. See the list.

Source document charset

Source document charset is set at Novelang startup, using `--source-charset` option (documented as an HTTP daemon feature).

Reading a source document in a charset which is not the one expected may result into incorrect character display, and even make the source document unreadable.

Charset mismatch can happen when working across different platforms with different default charsets. For Western European versions of Mac OS X, `MacRoman` is operating system's default charset, while for Western European versions of Microsoft Windows it is `Cp1250`.

In order to prevent various headaches, you must be aware of the charset of your source documents, and use a text editor which is explicit about the charset in use.

Recommended source document charset is `UTF-8`, which supports a wide range of characters.

Rendered document charset

Rendered document charset is set at Novelang startup, using `--rendering-charset` option (documented as an HTTP daemon feature).

The character set of the rendered document makes only sense for text-based formats like HTML. For HTML, Novelang will do its best to provide named HTML entities (making HTML source more readable). But, unless you need some special transcoding operation, `UTF-8` will always be great.

PDF don't care about rendered document charset as it uses Unicode internally. But rendered document may look wrong with a font that don't support its characters. Luckily, Novelang supports a preview of available fonts (with `/~fonts.pdf` pseudo-document).

Characters in the Novelang grammar

The characters recognized by Novelang are hardcoded in its guts. While Novelang reads and writes a lot of charsets, only some of the characters in this charset are supported. In order to keep the Novelang grammar meaningful, it's not possible to admit any character, especially symbols and punctuation signs. Letters, like Roman ones with diacritics, will be added on a case-by-case basis.

By now, Novelang supports all letters of those languages: English, French, Hungarian.

Additional resources about Unicode

— A brief introduction by Joel Spolsky.
— A longer article.

Custom stylesheets

It's easy to customize rendering of PDF, HTML and plain text because Novelang relies on FO (Formatting Objects) stylesheets.

Novelang looks for stylesheets in that order:
- In the directories set by `--style-dirs` option at startup.
- Or in a `style` directory right under the project directory (from where Novelang was launched), if the `--style-dirs` option was not set.
- Finally, inside Novelang's jars files under the `/style` directory.

By default Novelang attempts to render final document using the stylesheet with the name of corresponding format. Otherwise, it uses a default, built-in stylesheet.

```
MIME type extension                 .pdf     .html
Corresponding default stylesheet     pdf.xsl  html.xsl
```

After launching Novelang HTTP daemon, you can use the stylesheet query parameter to override any other stylesheet name:

```
http://localhost:8080/chapter-1.html?stylesheet=html-beautiful.xsl
```

Stylesheets may be defined for a whole Opus as explained later, see the "mapstylesheet" command.

eXtensible Stylesheet Language

The stylesheets are written in XSL/FO, which stands for eXtensible Stylesheet Language/ Formatting Objects. Both are standards developed by the W3C (World Wide Web Consortium). The reference documentation is here.

FO may look complex, because typesetting is inherently complex, and because of the lack of synthetic documentation. So you may be interested by this document explaining importants FO basics.

You'll find valuable tutorials on ZVon, Webucator and Dave Pawson's site.

XSL reuse

Novelang supports stylesheet reuse with standard `xsl:import` command. You can reuse Novelang's bundled stylesheets:

```
general-punctuation.xsl
punctuation-FR.xsl
punctuation-US-EN.xsl
default-pdf.xsl
default-html.xsl
```

Character entities

Novelang stylesheets support inclusion of character entities. This means, you can include definition of characters which can no be typed verbatim in the stylesheet, like the non-breaking space.

Such a definition looks like this:

```
<!ENTITY nbsp   "&#160;" >
```

So inside the stylesheet you just have to type "`&nbsp;`" instead of "`&#160;`".

Novelang comes bundled with those files. Here is how to refer them (it's quite verbose as XML always is):

```
<!DOCTYPE doctype [

  <!ENTITY % ISOnum PUBLIC
      "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML"
      "ISOnum.pen"
  >
  %ISOnum;

  <!ENTITY % ISOpub PUBLIC
      "ISO 8879:1986//ENTITIES Publishing//EN//XML"
      "ISOpub.pen"
  >
  %ISOpub;

  <!ENTITY % ISOlat1 PUBLIC
      "ISO 8879:1986//ENTITIES Added Latin 1//EN//XML"
      "ISOlat1.pen"
  >
  %ISOlat1;

]>
```

For HTML documents, those entities are automatically HTML-escaped when their system name starts with `ISO 8879:1986//ENTITIES` as above.

Other functions

Java developers can add functions on their own in order to augment standard set of functions available from an XSL stylesheet. By now there is one, `numberAsText` which transforms a number into its textual equivalent. For example, number “43” will become “forty-three”.

In addition to standard namespace declarations, the stylesheet must contain the `xalan` and `nlx` namespaces like below:

```
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:n="http://novelang.org/book-xml/1.0"
    xmlns:xalan="http://xml.apache.org/xalan"
    xmlns:nlx="xalan://org.novelang.rendering.xslt.Numbering"
>
```

Here is how to call the `numberAsText` function:

```
<xsl:value-of select="nlx:numberAsText(43,'EN','capital')" />
```

The first parameter is the number itself (could be a standard XSL function like `position()` as well). Currently the number must be included in the 0-50 range.

The second parameter is the locale. Currently only `EN` and `FR`.

The third parameter is the case. Currently `lower`, `upper` and `capital` are supported.

Look at the complete example in `samples/numbering/numbering.xsl` in the Novelang distribution.

Fonts

Novelang PDF generation relies on Apache FOP, which support custom fonts. Every PDF reader basically supports 14 fonts: serif, sans-serif, monospace, and some symbol fonts. Non-symbol fonts all come in four flavor: normal, bold, italic, and bold-italic. Not that much fun! So users are encouraged to set their own fonts.

Font files

Those fonts must be True Type fonts in `.ttf` files.

Requesting a manual setting of requested fonts may seem an unnecessary burden. On the other hand, people using publishing tools often complain about a missing font or a broken one, when importing documents created on another computer. By making fonts a part of a Novelang project (with same strategy as for content and stylesheets) we expect you to avoid those annoyances.

Font directories

If there is a directory named `fonts` right under your project root, it is automatically recognized as the fonts directory and Novelang attempts to load every file ending by `.ttf` as a font. This doesn't include subdirectories.

You can specify alternate font directories by setting the `--font-dirs` option at startup. Directory names are whitespace-separated like this:

```
--font-dirs fonts/some-fonts fonts/other-fonts
```

Font list

You can see the list of registered fonts from your browser with following URL:

```
http://localhost:8080/~fonts.pdf
```

Turn it on

Inside a XSL-FO stylesheet you set the font-family of a block or an inline element with the `font-family` attribute. With the "Bitstream-Vera-Sans" correctly installed you'll declare something like this:

```
<fo:block
    font-family="Bitstream-Vera-Sans"
    ...
>
  ...
```

Font licensing

All fonts shipping with Novelang are licensed under the GNU Public License. As a consequence, if you create an electronic document embedding those fonts (precisely what happens with a PDF) your document becomes GPL-licensed. If this is not the wished behavior, consider using other fonts.

Hyphenation

Novelang PDF generation relies on Apache FOP, which supports hyphenation. Hyphenation is language-dependant and has very complex rules. Luckily those rules are already defined for many languages, unluckily they're made available under various licenses, most of all being not compabible with the GNU Public License. That's the reason why they are not shipped with Novelang.

Licensing

First you have to check yourself if the license is compatible with intended usage. Licenses are explained here.

Download

If there is no licensing issue then download and install the file (s) with your bare hands. All hyphenation files are available in a single archive here.

Download `offo-hyphenation.zip`, expand it, and pick the file corresponding to the language of your need in the `hyph` directory. Don't forget to copy the `hyphenation.dtd` file (otherwise it won't work).

Hyphenation directory

If there is a directory named `hyphenation` right under your project root, it is automatically recognized as the directory containing hyphenation rules.

You can specify an alternate font directory by setting the `novelang.hyphenation.dir` system property.

Turn it on

Inside a XSL-FO stylesheet you set the language of a block with the `language` attribute and turn hyphenation on with the `hyphenate` attribute. You'll may have to tune hyphenation by setting `hyphenation-push-character-count` and `hyphenation-remain-character-count=` attributes. With the "FR" rules correctly installed you will declare something like this:

```
<fo:block
    language="FR"
    hyphenate="true"
    hyphenation-push-character-count="4"
    hyphenation-remain-character-count="4"
    ...
>
  ...
```

HTTP daemon

Novelang HTTP daemon runs as a Web server and displays documents in a Web browser.

It starts like this:

```
java -jar $NOVELANG_HOME/lib/Novelang-bootstrap-$VERSION.jar httpdaemon
 [options]
```

The `--port` option

The `--port` option sets the TCP port of which the daemon listens to:

```
--port=8083
```

Then URL for accessing documents becomes something like:

```
http://localhost:8083/mydocument.html
```

The `--serve-remotes` option

The `--serve-remotes` option enable serving document for other computers (computer with an IP address which is not 127.0.0.*). It is not recommended to activate this option because Novelang is not architectured yet to run as a Web server serving many concurrent requests.

```
--serve-remotes
```

Default value is `false`.

The `--content-root` option

The `--content-root` option sets the base directory to another value than current directory:

```
--content-root=../my-source/documents
```

The `--temporary-dir` option

The `--temporary-dir` option sets where Novelang writes its log files.

```
--log-temporary temporary-files
```

Default value is `$temporary$`.

The `--log-dir` option

The `--log-dir` option sets where Novelang writes its log files.

```
--log-dir logs
```

Default value is current directory (the value of `user.dir` system property).

The `--font-dirs` option

The `--font-dirs` option sets multiple directories where Novelang looks for fonts.

```
--font-dirs my/fonts-1 /Users/Shared/Fonts
```

The `--style-dirs` option

The `--style-dirs` option sets multiple directories where Novelang looks for stylesheets and related resources. See chapter about stylesheet for details.

```
--style-dirs my/styles-1 /Users/Shared/Fonts
```

The `--hyphenation-dir` option

The `--hyphenation-dir` option sets where Novelang should attempt to load hyphenation files from.

```
--hyphenation-dir my/directory
```

The `--source-charset` option

The `--source-charset` option sets the charset of source documents.

```
--source-charset MacRoman
```

Default value is `UTF-8`.

The `--rendering-charset` option

The `--rendering-charset` option sets the charset of rendered documents.

```
--rendering-charset iso-8859-2
```

Default value is `UTF-8`.

Directory listing

You can list the content of a directory by not giving any document name.

This will list every file ending by document source extension (currently “`.novella`” or “`.opus`”), including those in subdirectories. Every subdirectory also appear, even if it contains none of those files. Document sources become hyperlinks to their HTML form.

Here are samples of valid URLs for directory listings:

```
http://localhost:8080
http://localhost:8080/
http://localhost:8080/samples
http://localhost:8080/samples/
```

Because of a known Safari bug, Safari browsers get redirected to a fake page named `-.html` but the feature remains the same, however.

Batch document generator

The batch document generator is a batch tool generating one or more documents at once. A typical usage is from a shell script. Documents are requested with a path relative to the directory the batch generator was launched from.

If current directory contains a `hello.novella` file then following invocation will generate an HTML file named `output/hello.html`:

```
java -jar $NOVELANG_DIR/novelang-VERSION.jar generate
  /hello.html
```

The `--output-dir` option

The `--output-dir` option sets the output directory, where rendered documents are generated to.

```
java -jar $NOVELANG_DIR/novelang-VERSION.jar generate
  --output-dir generated/html /hello.html
```

Default value is `output`.

The `--content-root` option

The `--content-root` option sets the base directory to another value than current directory:

```
--content-root=../my-source/documents
```

The `--temporary-dir` option

The `--temporary-dir` option sets where Novelang writes its log files.

```
--log-temporary temporary-files
```

Default value is `$temporary$`.

The `--log-dir` option

The `--log-dir` option sets where Novelang writes its log files.

```
--log-dir logs
```

Default value is current directory (the value of `user.dir` system property).

The `--font-dirs` option

The `--font-dirs` option sets multiple directories where Novelang looks for fonts.

```
--font-dirs my/fonts-1 /Users/Shared/Fonts
```

The `--style-dirs` option

The `--style-dirs` option sets multiple directories where Novelang looks for stylesheets and related resources. See chapter about stylesheet for details.

```
--style-dirs my/styles-1 /Users/Shared/Fonts
```

The `--hyphenation-dir` option

The `--hyphenation-dir` option sets where Novelang should attempt to load hyphenation files from.

```
--hyphenation-dir my/directory
```

The `--source-charset` option

The `--source-charset` option sets the charset of source documents.

```
--source-charset MacRoman
```

Default value is `UTF-8`.

The `--rendering-charset` option

The `--rendering-charset` option sets the charset of rendered documents.

```
--rendering-charset iso-8859-2
```

Default value is `UTF-8`.

Level Exploder

The Level Exploder is a batch tool for generating multiple documents from the levels of one existing document. A typical usage is for slimming down a document that has become too big.

Document is requested with a path relative to the directory the level exploder was launched from.

If current directory contains a `big.novella` file with three levels named “one”, “two” and “three” then following invocation will generate three files named `output/one.novella`, `output/two.novella` and `output/three.novella`:

```
java -jar $NOVELANG_DIR/novelang-VERSION.jar explodelevels
  /big.novella
```

The `--output-dir` option

The `--output-dir` option sets the output directory, where rendered documents are generated to.

```
--output-dir exploded
```

Default value is `output`.

The `--content-root` option

The `--content-root` option sets the base directory to another value than current directory:

```
--content-root=../my-source/documents
```

The `--temporary-dir` option

The `--temporary-dir` option sets where Novelang writes its log files.

```
--log-temporary temporary-files
```

Default value is `$temporary$`.

The `--log-dir` option

The `--log-dir` option sets where Novelang writes its log files.

```
--log-dir logs
```

Default value is current directory (the value of `user.dir` system property).

The `--font-dirs` option

The `--font-dirs` option sets multiple directories where Novelang looks for fonts.

```
--font-dirs my/fonts-1 /Users/Shared/Fonts
```

The `--style-dirs` option

The `--style-dirs` option sets multiple directories where Novelang looks for stylesheets and related resources. See chapter about stylesheet for details.

```
--style-dirs my/styles-1 /Users/Shared/Fonts
```

The `--hyphenation-dir` option

The `--hyphenation-dir` option sets where Novelang should attempt to load hyphenation files from.

```
--hyphenation-dir my/directory
```

The `--source-charset` option

The `--source-charset` option sets the charset of source documents.

```
--source-charset MacRoman
```

Default value is `UTF-8`.

The `--rendering-charset` option

The `--rendering-charset` option sets the charset of rendered documents.

```
--rendering-charset iso-8859-2
```

Default value is `UTF-8`.

Supported characters

Escape name	Alias	Hex	Dec	Preview
«exclamation-mark»		u0021	0033	!
«double-quote»		u0022	0034	"
«number-sign»		u0023	0035	#
«dollar-sign»		u0024	0036	$
«percent-sign»		u0025	0037	%
«ampersand»	«amp»	u0026	0038	&
«apostrophe»		u0027	0039	'
«left-parenthesis»		u0028	0040	(
«right-parenthesis»		u0029	0041	)
«asterisk»		u002a	0042	*
«plus-sign»		u002b	0043	+
«comma»		u002c	0044	,
«hyphen-minus»		u002d	0045	-
«full-stop»		u002e	0046	.
«solidus»		u002f	0047	/
«digit-0»		u0030	0048	0

Escape name	Alias	Hex	Dec	Preview
«digit-1»		u0031	0049	1
«digit-2»		u0032	0050	2
«digit-3»		u0033	0051	3
«digit-4»		u0034	0052	4
«digit-5»		u0035	0053	5
«digit-6»		u0036	0054	6
«digit-7»		u0037	0055	7
«digit-8»		u0038	0056	8
«digit-9»		u0039	0057	9
«colon»		u003a	0058	:
«semicolon»		u003b	0059	;
«less-than-sign»	«lt»	u003c	0060	<
«equals-sign»		u003d	0061	=
«greater-than-sign»	«gt»	u003e	0062	>
«question-mark»		u003f	0063	?
«commercial-at»		u0040	0064	@
«latin-capital-letter-a»		u0041	0065	A
«latin-capital-letter-b»		u0042	0066	B
«latin-capital-letter-c»		u0043	0067	C
«latin-capital-letter-d»		u0044	0068	D
«latin-capital-letter-e»		u0045	0069	E
«latin-capital-letter-f»		u0046	0070	F
«latin-capital-letter-g»		u0047	0071	G
«latin-capital-letter-h»		u0048	0072	H
«latin-capital-letter-i»		u0049	0073	I
«latin-capital-letter-j»		u004a	0074	J
«latin-capital-letter-k»		u004b	0075	K
«latin-capital-letter-l»		u004c	0076	L
«latin-capital-letter-m»		u004d	0077	M
«latin-capital-letter-n»		u004e	0078	N
«latin-capital-letter-o»		u004f	0079	O
«latin-capital-letter-p»		u0050	0080	P
«latin-capital-letter-q»		u0051	0081	Q
«latin-capital-letter-r»		u0052	0082	R
«latin-capital-letter-s»		u0053	0083	S
«latin-capital-letter-t»		u0054	0084	T
«latin-capital-letter-u»		u0055	0085	U
«latin-capital-letter-v»		u0056	0086	V

Escape name	Alias	Hex	Dec	Preview
«latin-capital-letter-w»		u0057	0087	W
«latin-capital-letter-x»		u0058	0088	X
«latin-capital-letter-y»		u0059	0089	Y
«latin-capital-letter-z»		u005a	0090	Z
«left-square-bracket»		u005b	0091	[
«reverse-solidus»		u005c	0092	\
«right-square-bracket»		u005d	0093	]
«circumflex-accent»		u005e	0094	^
«low-line»		u005f	0095	_
«grave-accent»		u0060	0096	`
«latin-small-letter-a»		u0061	0097	a
«latin-small-letter-b»		u0062	0098	b
«latin-small-letter-c»		u0063	0099	c
«latin-small-letter-d»		u0064	0100	d
«latin-small-letter-e»		u0065	0101	e
«latin-small-letter-f»		u0066	0102	f
«latin-small-letter-g»		u0067	0103	g
«latin-small-letter-h»		u0068	0104	h
«latin-small-letter-i»		u0069	0105	i
«latin-small-letter-j»		u006a	0106	j
«latin-small-letter-k»		u006b	0107	k
«latin-small-letter-l»		u006c	0108	l
«latin-small-letter-m»		u006d	0109	m
«latin-small-letter-n»		u006e	0110	n
«latin-small-letter-o»		u006f	0111	o
«latin-small-letter-p»		u0070	0112	p
«latin-small-letter-q»		u0071	0113	q
«latin-small-letter-r»		u0072	0114	r
«latin-small-letter-s»		u0073	0115	s
«latin-small-letter-t»		u0074	0116	t
«latin-small-letter-u»		u0075	0117	u
«latin-small-letter-v»		u0076	0118	v
«latin-small-letter-w»		u0077	0119	w
«latin-small-letter-x»		u0078	0120	x
«latin-small-letter-y»		u0079	0121	y
«latin-small-letter-z»		u007a	0122	z
«left-curly-bracket»		u007b	0123	{
«vertical-line»		u007c	0124	|

Escape name	Alias	Hex	Dec	Preview
«right-curly-bracket»		u007d	0125	}
«tilde»		u007e	0126	~
«section-sign»	«sect»	u00a7	0167	§
«copyright-sign»	«copy»	u00a9	0169	©
«left-pointing-double-angle-quotation-mark»	«laquo»	u00ab	0171	«
«registered-sign»	«reg»	u00ae	0174	®
«degree-sign»	«deg»	u00b0	0176	°
«right-pointing-double-angle-quotation-mark»	«raquo»	u00bb	0187	»
«latin-capital-letter-a-with-grave»	«Agrave»	u00c0	0192	À
«latin-capital-letter-a-with-acute»	«Acute»	u00c1	0193	Á
«latin-capital-letter-a-with-circumflex»	«Acirc»	u00c2	0194	Â
«latin-capital-letter-a-with-diaeresis»	«Auml»	u00c4	0196	Ä
«latin-capital-letter-ae»	«AElig»	u00c6	0198	Æ
«latin-capital-letter-c-with-cedilla»	«Ccedil»	u00c7	0199	Ç
«latin-capital-letter-e-with-grave»	«Egrave»	u00c8	0200	È
«latin-capital-letter-e-with-acute»	«Ecute»	u00c9	0201	É
«latin-capital-letter-e-with-circumflex»	«Ecirc»	u00ca	0202	Ê
«latin-capital-letter-e-with-diaeresis»	«Euml»	u00cb	0203	Ë
«latin-capital-letter-i-with-acute»		u00cd	0205	Í
«latin-capital-letter-i-with-circumflex»	«Icirc»	u00ce	0206	Î
«latin-capital-letter-i-with-diaeresis»	«Iuml»	u00cf	0207	Ï
«latin-capital-letter-o-with-acute»		u00d3	0211	Ó
«latin-capital-letter-o-with-circumflex»	«Ocirc»	u00d4	0212	Ô
«latin-capital-letter-o-with-diaeresis»	«Ouml»	u00d6	0214	Ö
«multiplication-sign»	«times»	u00d7	0215	×
«latin-capital-letter-u-with-grave»	«Ugrave»	u00d9	0217	Ù
«latin-capital-letter-u-with-acute»	«Ucute»	u00da	0218	Ú
«latin-capital-letter-u-with-circumflex»	«Ucirc»	u00db	0219	Û
«latin-capital-letter-u-with-diaeresis»	«Uuml»	u00dc	0220	Ü
«latin-small-letter-a-with-grave»	«agrave»	u00e0	0224	à
«latin-small-letter-a-with-acute»	«acute»	u00e1	0225	á
«latin-small-letter-a-with-circumflex»	«acirc»	u00e2	0226	â
«latin-small-letter-a-with-diaeresis»	«auml»	u00e4	0228	ä
«latin-small-letter-ae»	«aelig»	u00e6	0230	æ
«latin-small-letter-c-with-cedilla»	«ccedil»	u00e7	0231	ç
«latin-small-letter-e-with-grave»	«egrave»	u00e8	0232	è

Escape name	Alias	Hex	Dec	Preview
«latin-small-letter-e-with-acute»	«ecute»	u00e9	0233	é
«latin-small-letter-e-with-circumflex»	«ecirc»	u00ea	0234	ê
«latin-small-letter-e-with-diaeresis»	«euml»	u00eb	0235	ë
«latin-small-letter-i-with-acute»		u00ed	0237	í
«latin-small-letter-i-with-circumflex»	«icirc»	u00ee	0238	î
«latin-small-letter-i-with-diaeresis»	«iuml»	u00ef	0239	ï
«latin-small-letter-o-with-acute»		u00f3	0243	ó
«latin-small-letter-o-with-circumflex»	«ocirc»	u00f4	0244	ô
«latin-small-letter-o-with-diaeresis»	«ouml»	u00f6	0246	ö
«latin-small-letter-u-with-grave»	«ugrave»	u00f9	0249	ù
«latin-small-letter-u-with-acute»	«ucute»	u00fa	0250	ú
«latin-small-letter-u-with-circumflex»	«ucirc»	u00fb	0251	û
«latin-small-letter-u-with-diaeresis»	«uuml»	u00fc	0252	ü
«latin-capital-letter-a-with-breve»	«Abreve»	u0102	0258	#
«latin-small-letter-a-with-breve»	«abreve»	u0103	0259	#
«latin-capital-letter-a-with-ogonek»		u0104	0260	#
«latin-small-letter-a-with-ogonek»		u0105	0261	#
«latin-capital-letter-c-with-acute»		u0106	0262	#
«latin-small-letter-c-with-acute»		u0107	0263	#
«latin-capital-letter-e-with-breve»	«Ebreve»	u0114	0276	#
«latin-small-letter-e-with-breve»	«Ebreve»	u0115	0277	#
«latin-capital-letter-e-with-ogonek»		u0118	0280	#
«latin-small-letter-e-with-ogonek»		u0119	0281	#
«latin-capital-letter-i-with-breve»	«Ibreve»	u012c	0300	#
«latin-small-letter-i-with-breve»	«ibreve»	u012d	0301	#
«latin-capital-letter-l-with-stroke»		u0141	0321	#
«latin-small-letter-l-with-stroke»		u0142	0322	#
«latin-capital-letter-n-with-acute»		u0143	0323	#
«latin-small-letter-n-with-acute»		u0144	0324	#
«latin-capital-letter-o-with-double-acute»		u0150	0336	#
«latin-small-letter-o-with-double-acute»		u0151	0337	#
«latin-capital-ligature-oe»	«OElig»	u0152	0338	Œ
«latin-small-ligature-oe»	«oelig»	u0153	0339	œ
«latin-capital-letter-s-with-acute»		u015a	0346	#
«latin-small-letter-s-with-acute»		u015b	0347	#
«latin-capital-letter-s-with-cedilla»	«Scedil»	u015e	0350	#
«latin-small-letter-s-with-cedilla»	«scedil»	u015f	0351	#

Escape name	Alias	Hex	Dec	Preview
«latin-capital-letter-t-with-cedilla»	«Tcedil»	u0162	0354	#
«latin-small-letter-t-with-cedilla»	«tcedil»	u0163	0355	#
«latin-capital-letter-u-with-breve»	«Ubreve»	u016c	0364	#
«latin-small-letter-u-with-breve»	«ubreve»	u016d	0365	#
«latin-capital-letter-u-with-double-acute»		u0170	0368	#
«latin-small-letter-u-with-double-acute»		u0171	0369	#
«latin-capital-letter-z-with-acute»		u0179	0377	#
«latin-small-letter-z-with-acute»		u017a	0378	#
«latin-capital-letter-z-with-dot-above»		u017b	0379	#
«latin-small-letter-z-with-dot-above»		u017c	0380	#
«latin-capital-letter-s-with-comma-below»		u0218	0536	#
«latin-small-letter-s-with-comma-below»		u0219	0537	#
«latin-capital-letter-t-with-comma-below»		u021a	0538	#
«latin-small-letter-t-with-comma-below»		u021b	0539	#
«greek-capital-letter-alpha»	«Alpha»	u0391	0913	#
«greek-capital-letter-beta»	«Beta»	u0392	0914	#
«greek-capital-letter-gamma»	«Gamma»	u0393	0915	#
«greek-capital-letter-delta»	«Delta»	u0394	0916	#
«greek-capital-letter-epsilon»	«Epsilon»	u0395	0917	#
«greek-capital-letter-zeta»	«Zeta»	u0396	0918	#
«greek-capital-letter-eta»	«Eta»	u0397	0919	#
«greek-capital-letter-theta»	«Theta»	u0398	0920	#
«greek-capital-letter-iota»	«Iota»	u0399	0921	#
«greek-capital-letter-kappa»	«Kappa»	u039a	0922	#
«greek-capital-letter-lambda»	«Lambda»	u039b	0923	#
«greek-capital-letter-mu»	«Mu»	u039c	0924	#
«greek-capital-letter-nu»	«Nu»	u039d	0925	#
«greek-capital-letter-xi»	«Xi»	u039e	0926	#
«greek-capital-letter-omicron»	«Omicron»	u039f	0927	#
«greek-capital-letter-pi»	«Pi»	u03a0	0928	#
«greek-capital-letter-rho»	«Rho»	u03a1	0929	#
«greek-capital-letter-sigma»	«Sigma»	u03a3	0931	#
«greek-capital-letter-tau»	«Tau»	u03a4	0932	#
«greek-capital-letter-upsilon»	«Upsilon»	u03a5	0933	#

Escape name	Alias	Hex	Dec	Preview
«greek-capital-letter-phi»	«Phi»	u03a6	0934	#
«greek-capital-letter-chi»	«Chi»	u03a7	0935	#
«greek-capital-letter-psi»	«Psi»	u03a8	0936	#
«greek-capital-letter-omega»	«Omega»	u03a9	0937	#
«greek-small-letter-alpha»	«alpha»	u03b1	0945	#
«greek-small-letter-beta»	«beta»	u03b2	0946	#
«greek-small-letter-gamma»	«gamma»	u03b3	0947	#
«greek-small-letter-delta»	«delta»	u03b4	0948	#
«greek-small-letter-epsilon»	«epsilon»	u03b5	0949	#
«greek-small-letter-zeta»	«zeta»	u03b6	0950	#
«greek-small-letter-eta»	«eta»	u03b7	0951	#
«greek-small-letter-theta»	«theta»	u03b8	0952	#
«greek-small-letter-iota»	«iota»	u03b9	0953	#
«greek-small-letter-kappa»	«kappa»	u03ba	0954	#
«greek-small-letter-lambda»	«lambda»	u03bb	0955	#
«greek-small-letter-mu»	«mu»	u03bc	0956	µ
«greek-small-letter-nu»	«nu»	u03bd	0957	#
«greek-small-letter-xi»	«xi»	u03be	0958	#
«greek-small-letter-omicron»	«omicron»	u03bf	0959	#
«greek-small-letter-pi»	«pi»	u03c0	0960	#
«greek-small-letter-rho»	«rho»	u03c1	0961	#
«greek-small-letter-final-sigma»	«sigmaf»	u03c2	0962	#
«greek-small-letter-sigma»	«sigma»	u03c3	0963	#
«greek-small-letter-tau»	«tau»	u03c4	0964	#
«greek-small-letter-upsilon»	«upsilon»	u03c5	0965	#
«greek-small-letter-phi»	«phi»	u03c6	0966	#
«greek-small-letter-chi»	«chi»	u03c7	0967	#
«greek-small-letter-psi»	«psi»	u03c8	0968	#
«greek-small-letter-omega»	«omega»	u03c9	0969	#
«right-single-quotation-mark»	«lsquo»	u2018	8216	‘
«left-single-quotation-mark»	«rsquo»	u2019	8217	’
«single-left-pointing-angle-quotation-mark»	«lsaquo»	u2039	8249	‹
«single-right-pointing-angle-quotation-mark»	«rsaquo»	u203a	8250	›
«euro-sign»	«euro»	u20ac	8364	€