Open Geneva User Manual

Interstellar Ventures

Saturday, 19 September 2015

Table of Contents

Open Geneva is an implementation of the *Geneva document preparation system* written in *Common Lisp*. This user manual describes the components of Open Geneva from a high level perspective and explains their operation by example. For a complete API documentation see the *Open Geneva API*.[1]

Open Geneva is divided into several subsystems, each implementing a different functionality of the system. For convenience, a "master system" is provided, which depends on every subsystem of Open Geneva. If you want to load and/or compile Open Geneva as a whole, then you may use the `open-geneva` system. The various subsystems are described in the sections below.

- 1. *Open Geneva API* (`open-geneva-api.html`)

1 Geneva: The Document API

At the core of Open Geneva are a set of constructors and readers used to programatically create and inspect Geneva documents. These functions are in the `geneva` package. These constructors verify the integrity of their arguments and their return values are normalized as defined in the *Geneva Document Specification*.[1]

There are three different kinds of constructors: The *document* constructor `make-document`, *document element* constructors (`make-pargraph` for instance) and *text token* constructors (`make-bold` etc.).

```
(defun make-birthday-invitation (date guest-name)
  (make-document
   (list
    (make-section
      '("Birthday Invitation")
     (list
      (make-paragraph
       `(,(make-bold (format nil "Hi ~a!" guest-name))))
      (make-paragraph
       `(,(format nil "You are invited to my birthday party on ~a. "
                  date)
         ,(make-italic "Bring your friends!")))))))))

(make-birthday-invitation "Friday" "John") → document
```

Example: Dynamically creating a document.

The readers `content-type` and `content-values` work on document elements as well as on text tokens and can be used to inspect the contents of a document. `Content-type` returns the type of its argument and `content-values` returns the components of it argument a seperate values.

```
(content-type (make-bold "foo")) → :BOLD
(content-type "bar") → :PLAIN ; Strings have a CONTENT-TYPE.
(content-values (make-section <title> <body>) → <title> <body>
```

Examples: Inspecting document contents.

A document is just a list of document elements. It can be traversed by the standard list manipulation functions.

```
;; Return list of element types used in document.
(defun document-features (document)
  (remove-duplicates
   (loop for element in document
         for type = (content-type element)
     if (eq type :section)
     then append (multiple-value-bind (title body)
                   (content-values element)
                 `(:section ,@(document-features body)))
     else collect (content-type element))))

(document-features (make-document
                    (make-paragraph '("foo"))
                    (make-paragraph '("bar"))))
  → (:PARAGRAPH)
```

Example: Traversing a document.

A document can be printed *readably* by the Common Lisp printer. The easiest way to (de)serialize a document is to use `read` and `print`.

```
(let ((document (make-document ...)))
  (equal document
         (read-from-string
          (prin1-to-string document))))
  → T
```

Example: (De)serializing a document.

The `geneva.macros` package provides macro counterparts of the element constructors and a readtable[2] `geneva.macros:syntax` which can come in handy when dynamically creating documents. Below is the "birthday invitation" example from above revisited using `geneva.macros`.

```
(in-readtable geneva.macros:syntax)

(defun make-birthday-invitation (date guest-name)
  (document
   (section ("Birthday invitation")
     (paragraph (make-bold (format nil "Hi ~a!" guest-name)))
     (paragraph
      (format nil "You are invited to my birthday party on ~a. "
              date)
      ;; Note the reader macro below.
      #i"Bring your friends!"))))
```

Example: Dynamically creating documents using `geneva.macros`.

- 1. *Geneva Document Specification* (`geneva-document.html`)
- 2. See *Named-Readtables* (`editor-hints.named-readtables`)

2 Geneva-mk2: Reading and Writing Mk2 Files

Mk2[1] is a human readable serialization format for Geneva documents. Open Geneva implements the Mk2 markup language in the `geneva.mk2` package. Geneva documents can be read from and printed as Mk2 using `read-mk2` and `print-mk2`.

Note that an Mk2 file is a precise representation of a Geneva document. The following holds true:

```
(let ((document (make-document ...)))
  (equal document
         (read-mk2 (with-output-to-string (out)
                     (print-mk2 document out)))))
  → T
```

- 1. *The Mk2 Markup Language* (`mk2.html`)

3 Rendering Geneva Documents

Open Geneva supports rendering Geneva documents as plain text, HTML and LaTeX. The implementing functions can be loaded as

the `geneva-plaintext`, `geneva-html` and `geneva-latex` systems respectively.

3.1 Common Rendering Interface

The various rendering systems share a common subset of their interface.

— Function: **render-plain-text** | **render-html** | **render-latex** *document* `&key` *stream title author date index-p index-caption index-headers-p* `&allow-other-keys`

Arguments and Values:

document—a Geneva *document*.

stream—a *character stream*. The default is *standard output*.

title—a *string*.

author—a *string*.

date—a *string*.

index-p—a *generalized boolean*. The default is *true*.

index-caption—a *string*. The default is `"Table of Contents"`.

index-headers-p—a *generalized boolean*. The default is *true*.

Description:

Renders *document* to *stream*. The document rendering can optionally be prepended by a title section and a section index. *Title*, *author* and *date* are used in the title section. *Index-caption* can be supplied to customize the heading of the section index. If *index-p* is *false* the section index will be omitted. Section headers will be enumerated unless *index-headers-p* is *false*.

Exceptional Situations:

If *document* is not a valid Geneva *document* an *error* of *type* `type-error` is signaled.

4 Geneva-cl: Compiling Geneva Documents from Common Lisp On-Line Documentation

The `geneva.common-lisp` package provides a function `api-document` which can be used to compile Geneva documents from Common Lisp on-line documentation. Its usage is quite simple and can be explained by example:

```
(defpackage foo
  (:documentation "Foo is a demo package.")
  (:use :cl)
  (:export :bar))

(defun foo:bar (x) "{bar} is a _NO-OP_." x)

(api-document :foo)
 → ((:SECTION ("foo")
     ((:PARAGRAPH ("Foo is a demo package."))
      (:SECTION ("bar")
       ((:PARAGRAPH ("- Function: " (:BOLD "bar") " " (:ITALIC "x")))
        (:PARAGRAPH ((:FIXED-WIDTH "bar") " is a "
                     (:ITALIC "NO-OP") "."))))))))
```

Creating an API document from a package.

Note that documentation strings are parsed as *Mk2* files using `read-mk2`.