NAME

unroff – programmable, extensible troff translator

SYNOPSIS

unroff [**–f***format*] [**–m***package*] [**–h***heapsize*] [**–C**] [**–t**] [*file* | *option...*]

OVERVIEW

unroff reads and parses documents with embedded troff markup and translates them to a different format—typically to a different markup language such as SGML. The actual output format is not hard-wired into *unroff*; instead, the translation is performed by a set of user-supplied rules and functions written in the *Scheme* programming language. *unroff* employs the Extension Language Kit *Elk* to achieve programmability based on the Scheme language: a fully-functional Scheme interpreter is embedded in the translator.

The documents that can be processed by *unroff* are not restricted to a specific troff macro set. Translation rules for a new macro package can be added by supplying a set of corresponding Scheme procedures (a "back-end"). Predefined sets of such procedures exist for a number of combinations of target language and troff macro package: *unroff* 1.0 supports translation to the "Hypertext Markup Language" (HTML) version 2.0 for the **–man** and **–ms** macro packages as well as "bare" troff (see **unroff-html**(1), **unroff-html-man**(1), and **unroff-html-ms**(1) for a description).

Unlike conventional troff conversion tools, *unroff* includes a full troff parser and can therefore handle user-defined macros, strings, and number registers, nested if-else requests (with text blocks enclosed by '\{' and '\}' escape sequences), arbitrary fonts and font positions, troff "copy mode", low-level formatting requests such as '\l' and '\h', and the subtle differences between request and macro invocations that are inherent in the troff processing model. *unroff* has adopted a number of troff extensions introduced by *groff*, among them long names for macros, strings, number registers, and special characters, and the '\$@' and '\$*' escape sequences.

unroff interprets its input stream as a sequence of "events". Events include the invocation of a troff request or macro, the use of a troff escape sequence or special character, a troff string or number register reference, end of sentence, start of a new input file, and so on. For each event encountered *unroff* invokes a Scheme procedure associated with that event. Some types of events require a procedure that returns a string (or an object that can be coerced into a string), which is then interpolated into the input or output stream; for other types of events, the event procedures are just called for their side-effects.

The set of Scheme procedures to be used by *unroff* is determined by the output format and the name of the troff macro package. In addition, users can supply event procedures for their own macro definitions (or replace existing ones) in form of a simple Scheme program passed to *unroff* along with the troff input files; Scheme code can even be directly embedded in the troff input as described below.

The full capabilities of *unroff* and the Scheme primitives required to write extensions or support for new output formats are described in the *Unroff Programmer's Manual*.

GENERIC OPTIONS

–f*format*

Specifies the output format into which the troff input files are translated. If no **–f** option is given, a default output format is used (for *unroff* version 1.0 the default is **–f***html*). This default can be overridden by setting the **UNROFF_FORMAT** environment variable.

–m*name*

Specifies the name of the macro package that would be used by ordinary troff to typeset the document. In contrast to troff *unroff* does not actually load the macro package. Instead, the specified name–in combination with the specified output format–selects a set of Scheme files providing the procedure definitions that control the translation process (see **FILES** below). Therefore a corresponding **tmac** file need not exist for a given **–m** option.

–h*heapsize*

This option can be used to specify a non-standard heap size (in Kbytes) for the Scheme interpreter included in *unroff*; see **elk**(1).

−C Enables troff compatibility mode. In compatibility mode certain *groff* extensions such as long names are not recognized.

−t Enables test mode. Instead of processing troff input files, *unroff* enters an interactive Scheme top-level. This can be useful to interactively experiment with the Scheme primitives defined by *unroff* or to test or debug user-defined Scheme procedures.

KEYWORD/VALUE OPTIONS

In addition to the generic options, a set of output-format-specific options can be set from the command line and from within troff and Scheme input files. When specified on the command line, these options have the form

option=*value*

where the format of *value* depends on the *type* of the option. For example, most output formats defines an option **document** whose value is used as a prefix for all output files created during the translation. The option is assigned a value by specifying a token such as

document=thesis

on the command line. This option's value is interpreted as a plain string, i. e. its type is **string**.

The Scheme back-ends and user-supplied extensions can define their own option types, but at least the following types are recognized:

integer	the option value is composed of an optional sign and an (arbitrary) string of digits
boolean	the option value must either be the character 1 (true) or the character 0 (false)
character	a single character must be specified as the option value
string	an arbitrary string of characters can be specified
dynstring	"dynamic string"; the option value is either

string	to assign a string to the option in the normal way, or
+*string*	to append the characters after the plus sign to the option's current value, or
−*string*	to remove the characters after the minus sign from the option's current value.

These extension-specific options must appear after the generic *unroff* options and may be mixed with the file name arguments. As the option assignments and specified input files are processed in order, the value given for an option is in effect for all the input files that appear on the command line to the right of the option.

The exact set of keyword/value options is determined by the Scheme code loaded for a given combination of output format and macro package name and is described in the corresponding manuals. The following few options can always be set, regardless of the actual output format:

include-files (boolean)
If true, **.so** requests are executed by *unroff* in the normal way (that is, the named input file is read and parsed), otherwise **.so** requests are ignored. The default value is 1.

if-true (dynstring)
the specified characters are assigned to (appended to, removed from) the set of one-character conditions that are regarded as true by the **.if** and **.ie** requests. The default value is "to".

if-false (dynstring)
like **if-true**; specifies the one-character conditions regarded as false. The default value is "ne".

FILES

INPUT FILES

On startup, *unroff* loads the Scheme source files that control the translation process. All these files are loaded from subdirectories of a site-specific "library directory", typically something like **/usr/local/lib/unroff**. The directory is usually chosen by the system administrator when installing the software and can be overridden by setting the **UNROFF_DIR** environment variable. The path names mentioned in the following are relative to this library directory.

The first Scheme file loaded is **scm/troff.scm** which contains basic definitions such as the built-in options and option types, implementations for troff requests that are not output-format specific, and utility functions to be used by the back-ends or by user-supplied extensions. Next, the file **scm/***format***/common.scm** is loaded, where *format* is the value of the option **–f** as given on the command line (or its default value). The file implements the translation of the basic troff requests, escape sequences, and special characters, etc. The code dealing with macro invocations is loaded from **scm/***format***/***package***.scm** where *package* is the value of the option **–m** with the letter 'm' prepended.

Finally, the file **.unroff** is loaded from the caller's home directory if present. Arbitrary Scheme code can be placed in this initialization file. It is typically used to assign values to package-specific keyword/value options according to the user's preferences (by means of the *set-option!* Scheme primitive as explained in the Programmer's Manual).

When the initial files have been loaded, any troff input files specified in the command line are read and parsed. The special file name '**–**' can be used to indicate standard input (usually in combination with ordinary file names). If no file name is given, *unroff* reads from standard input.

In addition to troff input files, file containing Scheme code can be mentioned in the command line. Scheme files (which by convention end in **.scm**) are loaded into the Scheme interpreter and usually contain used-defined Scheme procedures to translate specific macros or to replace existing procedures, or other user-supplied extensions of any kind. Scheme files named in the command line (or loaded explicitly from within other files) are resolved against the directory **scm/misc/** which may hold site-specific extensions or other supplementary packages. troff files and Scheme files can be mixed freely in the command line.

OUTPUT FILES

Whether *unroff* sends its output to standard output or produces one or more output files is not hard-wired but determined by the combination of output format and macro package. Generally, if no troff input files are specified, output is directed to standard output, but this rule is not mandatory and may be overridden by specific back-ends. The **document** option is usually honored, although other rules may be employed to determine the names of output files (for example, the extension that implements **–man** for a given output format may derive the name of the output file for a manual page from the input file name; see **unroff-html-man**(1)).

If *unroff* is interrupted or quits early, any output files produced so far may be incomplete or may contain wrong or inconsistent data, because several passes may be required to complete an output file (for example, to resolve cross references between a set of files), or because an output file is not necessarily produced as a whole, but *unroff* may work on several files simultaneously.

EXAMPLES

To translate a troff document composed of two files and written with the "ms" macro package to HTML 2.0, *unroff* might be called like this:

```
unroff –fhtml –ms doc.tr doc.tr
```

Two options specific to the combination of **–fhtml** and **–ms** might be added to specify a prefix for output files and to have the resulting output split into separate files after each section (see **unroff-html-ms**(1)):

```
unroff –fhtml –ms document=out/ split=1 doc.tr doc.tr
```

Additional features may be loaded from Scheme files specified in the command line, e. g. **hyper.scm** which implements general Hypertext requests (and gets loaded from **scm/misc/**) and a user-supplied file in the current directory providing translation rules for user-defined troff macros:

```
unroff –fhtml –ms document=out/ split=1 hyper.scm doc.scm\
    doc.tr doc.tr
```

TROFF SUPPORT AND EXTENSIONS

As *unroff* translates troff input into another language rather than typesetting the text in the usual way, its processing model necessarily differs from that of conventional troff. For a detailed description refer to the

Programmer's Manual.

In brief, *unroff* copies characters from input to output, optionally performing target-language-specific character translations. For each request or macro invocation, string or number register reference, special character, escape sequence, sentence end, or **eqn**(1) inline equation encountered in the input stream, *unroff* checks whether an "event value" has been specified by the Scheme code (user-supplied or part of the back-end). An event value is either a plain string, which is then treated as if it had been part of the input stream, or a Scheme procedure, which is then invoked and must in turn return a string. The Scheme procedures are passed arguments, e. g. the macro or request arguments in case of a procedure attached to a macro or request, or an escape sequence argument for functions such as '\f' or '\w'.

If no event value has been associated with a particular macro, string, or number register, *unroff* checks whether a definition has been supplied in the normal way, i. e. by means of **.de**, **.ds**, or **.nr**. In this case, the value of the macro, string, or register is interpolated as done by ordinary troff. If no definition can be found, a fallback definition is looked up as a last resort; and if everything fails, a warning is printed and the event is ignored. Similarly, event procedures are invoked at end of input line, when an input file is opened or closed, at program start and termination, and for each option specified in the command line; but these procedures are called solely for their side-effects (i. e. the return values are ignored).

Most Scheme procedures just emit the target language's representation of the event with which they are associated. Other procedures perform various kinds of bookkeeping; the procedure associated with the **.de** request, for example, puts the text following aside for later expansion, and the event procedures attached to the requests **.ds** and **.nr** and to the escape sequences '*' and '\n' implement troff strings and number registers. This way, even basic troff functions need not be hard-wired and can be altered or replaced freely without recompiling *unroff*.

The rule that an event value associated with a macro has precedence over the actual macro definition accommodates higher-level, structure-oriented target languages (such as SGML). While the micro-formatting contained in a typical **–ms** macro definition, for example, makes sense to an ordinary typesetting program, it is usually impossible to infer the macro's *structural* function from it (new paragraph, quotation, etc.). On the other hand, troff documents often define a few additional, simple macros that just serve as an abbreviation for a sequence of predefined macros; in this case event procedures need not specified, as *unroff* will then perform normal macro expansion.

unroff usually takes care to not rescan the characters returned by event procedures as if their results had been normal input, because most event procedures already return code in the target language rather than troff input that can be rescanned. This, however, cannot always be avoided; for example, if a troff string reference occurs at macro definition time (because '*' is used rather than '\\*'), the string value ends up in the macro body and will still be rescanned when the macro is invoked. A few other pitfalls caused by differences in the processing models of troff and *unroff* are listed in the BUGS section below.

The scaling performed for the usual troff scale indicators can be manipulated by a calling a Scheme primitive from within the Scheme code implementing a particular back-end.

NEW TROFF REQUESTS

To aid transparent output of code in the target language and evaluation of inline Scheme code, *unroff* supports two new requests and two extensions to the **.ig** (ignore input lines) troff request.

If **.ig** is called with the symbol **>>** as its first argument, all input lines up to (but not including) the terminating **.>>** are sent to the current output file. Example: when translating to the Hypertext Markup Language, the construct could be used to emit literal HTML code like this:

```
.ig >>
<address>
Bart Simpson<br>
Springfield
</address>
.>>
```

To produce a single line of output, the new request **.>>** can be used as in this HTML example:

```
.>> "<code>result = i+1;</code>"
```

If the **.ig** request is called with the argument **##,** everything up to the terminating **.##** is passed to the Scheme interpreter for evaluation. This allows users to embed Scheme code in a troff document which is executed when the document is processed by *unroff*. One use of this construct is to provide a Scheme event procedure for a user-defined macro by placing the corresponding Scheme definition in the same source file right below the troff macro definition. Similarly, the request **.##** can be used to evaluate a short S-expression; all arguments to the request are concatenated and then passed to the Scheme interpreter.

Note that inline Scheme code is a potentially dangerous feature, as a document received by someone else may contain embedded code that does something unexpected when the file is processed by *unroff* (but it is probably not more dangerous than the standard troff **.pi** request or the **.sy** request of *ditroff*).

unroff defines the following new read-only number registers:

.U This register always expand to 1. It can be used by macros to determine whether the document is being processed by *unroff*.

.C Expands to 1 if troff compatibility mode has been enabled by using the option **−C**, to 0 otherwise.

The following new escape sequences are available in a macro body during macro expansion:

$0 The name of the current macro.

$* The concatenation of all arguments, separated by spaces.

$@ The concatenation of all arguments, separated by spaces, and with each argument enclosed by double quotes.

The names of strings, macros, number registers, and fonts may be of any length. As in *groff*, square brackets can be used for names of arbitrary length:

```
\f[font]  \*[string]  \n[numreg]  ...
```

There is no limit on the number of macro arguments, and the following syntax can be used to reference the 10th, 11th, etc. macro argument:

```
\$(12  \$[12]  \$[123]
```

Unless troff compatibility mode has been enabled, the arguments to the *groff*-specific escape sequences '\A', '\C', '\L', '\N', '\R', '\V', '\Y', and '\Z' are recognized and parsed, so that event procedures can be implemented correctly for these escape sequences.

SEE ALSO

unroff-html(1), **unroff-html-man**(1), **unroff-html-ms**(1);
troff(1), **groff**(1); **elk**(1).

Unroff Programmer's Manual.

http://www.informatik.uni-bremen.de/˜net/unroff

AUTHOR

Oliver Laumann, net@cs.tu-berlin.de

BUGS

A number of low-level formatting features of troff (such as the absolute position indicator in numerical expressions) are not yet supported by *unroff* version 1.0, which is not critical for higher-level, structure-oriented target languages such as the Hypertext Markup Language.

Diversions are not supported, although specific back-ends are free to add this functionality.

Special characters are not treated right in certain contexts; in particular, special characters may not be used in place of plain characters where the characters act as some kind of delimiter as in

```
.if \(bsfoo\(bsbar\(bs ...
```

Spaces in an **.if** condition do not work; e. g. the following fails:

.if ' ' ' ...

Conditional input is subject to string and number register expansion even if the corresponding if-condition evaluates to false.

There are no number register formats, i. e. the request **.af** does not work.

The set of punctuation marks that indicate end of sentence should be configurable.

Empty input lines and leading space should trigger a special event, so that their break semantics can be implemented correctly.

A comment in a line by itself currently does not generate a blank line.