

2. Running Sparser

Depending on one's licensing agreement, Sparser is distributed either as a Macintosh application or as a directory tree of compiled or source files in Common Lisp (following the ANSI X3J13 standard as described in the second edition of Steele's *Common Lisp*). The primary platform for Sparser is the Macintosh running version 2.0 of Macintosh Common Lisp, and there are interactive tools that run on that platform only. Versions of Sparser are available to run under Unix using Common Lisps developed by Lucid, Franz, and CMU. Ports to other platforms and versions of Common Lisp can be accommodated.

To use Sparser, the application must be launched (using whatever means are appropriate for your operating system, such as double-clicking the icon when using a Macintosh), or, alternatively, Sparser's files must be loaded into a running Lisp, perhaps one already containing parts of your own systems. In both cases, once Sparser is ready you will see the following banner announcements, whereupon you will be at the toplevel of your normal Lisp read-eval-print loop, and can call Sparser's functions or your own as you would normally.

```
Welcome to the Sparser natural language analysis system
copyright David D. McDonald 1991-2005,2010-2014 all rights reserved.
```

```
Type (in-package :sparser) to use Sparser symbols directly.
```

In some Macintosh versions of Sparser, there will also be additional, Sparser-specific menus on the menu bar that will let you run most of the common operations from menus and to examine its results and extend parts of its grammar in a special workbench. These are described in §8.2.

2.1 Calling Sparser

Sparser is called as a function that takes a sequence of characters as its input and produces a populated chart and a sequence of triggered actions as its output. The input can be provided as either a string or a file.

When Sparser returns, it leaves the chart and all of the globally bound state variables unchanged from the values they had at the moment it completed its analysis. They will not be re-initialized until the next call to analyze a text is made, and so can be examined by the user or the user's code at leisure.

The two functions below are the only entry points for running Sparser. When one of them is called, the entire input is processed without stopping or looking for mouse or keyboard activity by the user, except as may be provided for in any user-defined code or traces executing at designated hooks (see §7).

`analyze-text-from-string (string)` function

Runs Sparser on the indicated string. Returns the keyword `:analysis-completed` when the entire string has been processed.

`analyze-text-from-file (file)` function

Runs the parser on the indicated file. Returns the keyword `:analysis-completed` when all of the characters in the file have been processed. The ‘file’ argument may be either a Lisp pathname or a string that can be converted to a pathname by the Lisp function `pathname`. If the file does not exist, the error will be signaled by the Lisp function `open`.¹

Both functions employ a common core routine (§6.0.1) and differ only in how they set up the character buffer (§6.2). After initializing the tokenizer and the chart, this core routine calls the function “chart-based-analysis” to initiate the analysis process. Since the subroutines that make up Sparser’s control algorithms are all tail-recursive, this initiating function is the only one that is certain to be present on the stack at a breakpoint. The overall analysis algorithm can be varied within certain limits as described in the section on parameterizations (§6.1).

2.1.1 Managing file input

When the input is a file, the following pointer is kept. As its name suggests, it keeps track of the Lisp stream that was created when the input file was opened. Calls against this stream supply the sequence of characters to be analyzed; see §6.2.

`*open-stream-of-source-characters*` symbol (defvar)

This stream will be closed automatically when `analyze-text-from-file` finishes. However, if the process terminates unusually, for instances because of some Lisp error, then the file will remain open and this symbol will continue to point to it. In some Lisps there is a limit on the number of files that can be open simultaneously, so this can become problematic. This symbol is checked at the start of `Analyze-text-from-file`, and if the old stream is still active it is closed before a new one is opened. It can also be closed by this function, which also sets the symbol to `nil`.

`close-character-source-file ()` function

Closes the Lisp stream that was opened by the last, unfinished call to `analyze-text-from-file`, and sets the symbol that points to this stream to `nil`. If the symbol is already `nil` at the time the call is made, no special action is taken.

¹ A semi-colon in a namestring passed to `Analyze-text-from-file` will have a different effect than is described in the Common Lisp manual under the discussion of ‘Logical Pathnames’. It does signal that the (single) symbol to its left is to be interpreted as a logical directory, but the system of logical directories it utilizes is internal to Sparser’s implementation; it was an outgrowth of the logical directory facility originally provided with Macintosh Common Lisp.

2.1.2 Terminating early

When using Sparser to search through a large text, it can be useful to be able to stop further processing before the entire text has been analyzed. The following function can be included in a user's code for this purpose:

```
terminate-chart-level-process ( ) function
```

Executes a Lisp “throw” that forces the toplevel function call that initiated the parse to immediately return. If the text is being taken from a file, the stream to that file is closed.

2.3 Lisp Packages

The Lisp symbols in Sparser are all in one of five packages. In particular, all of the functions and global symbols mentioned in this documentation are in the SPARSER package—when working with the parser you should probably move to that package to avoid any confusion. The only other case is a few symbols used in loading the system (§2.2) which are in the CL-USER package (“Common Lisp User”, following the ANSI X3J13 standard as described in the second edition of Steele's *Common Lisp*).

```
"SPARSER" :nickname nil :includes ("COMMON-LISP") package
```

All function names and global symbols are in this package. In particular, this includes the functions and macros used to define rules. Any file of rule forms should be in this package. If Sparser is being run on a Macintosh under MCL, the package CCL will also be included.

```
"WORD" :nickname nil :includes ( ) package
```

The symbols in this package are constructed automatically by the system to point to word objects when they are defined (§3.1.2). They are central to the the tokenizer's procedure for indexing from character strings to word objects. For users they are convenient for accessing word objects interactively or for referencing them in code.

```
"CATEGORY" :nickname nil :includes ( ) package
```

The symbols in this package are constructed automatically by the system to point to category objects when are defined (§3.3). They have no run-time role, but are used in the notation for phrase structure rules and are useful for interactive access or for referencing them in code.

```
"RULE"           :nickname nil      :includes ()           package
```

The symbols in this package are constructed automatically by the system to point to phrase structure rules when they are defined (§4.1.4). They have no run-time role, and are only used for convenient interactive access or references in code. All of these symbols are gensyms, i.e. their print names consist of the base “PSR-” followed by an integer.

```
"BRACKETS"      nickname nil      :includes ()           package
```

The symbols in this package are constructed automatically by the system to point to bracket objects when they are defined. They have no run-time role, but are used in the notation for bracket assignments, and are useful for interactive access or referencing brackets in code.

The names of these packages are not mentioned in Sparser’s source code except at the points where they are defined. A set of constant symbols, bound to these packages, are used instead. These are

```
*sparser-source-package*,
*word-package*,
*category-package*,
*phrase-structure-rule-package*, and
*brackets-package*.
```

The constants are themselves in the `sparser` package as are all other source symbols. Arrangements can be made to have Sparser installed with alternative spellings of the package names to avoid conflicts with the user’s package set.