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 anouncements, 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.

2.2 Installation

Depending on one's licensing agreement, Sparser is distributed either as a Macintosh application or as a directory tree of compiled or source files. If you have the application version then you simply install it in whatever directory you like following the instructions provided with the delivery. The procedures for installating the directory tree and creating your own application image by loading the files are somewhat more involved and are described in this section.

2.2.1 Directory Structure

Sparser is organized as a deep directory tree of approximately a thousand files. It is loaded under the direction of a master load file which takes parameters to indicate where the files are stored and which subsets of them should be loaded. The names of the files and their directory structure may not be changed, though the location of the root directory "Sparser" does not matter. In this section we describe enough of the this directory structure to let you make changes you may want in configuration files or distributed grammar rules.

Your file distribution will be rooted in a directory called "Sparser". (Note the capital "S"; Macintosh filenames are case-sensitive, and permit a wide range of characters including 'space', and Sparser uses these abilities.) The directories that you may need to know about just below "Sparser" are shown below.

```
> Sparser
>  configuration
>    launch
>    load
>  code
>    f
>  grammar
>    ...
>  load Sparser
```

"configuration" is a directory containing the two files "launch" and "load" that are described in §6.1.

"code" is a directory with one daughter: "f" which contains a set of directories and multiple levels of subdirectories containing the compile code ("fasl" files) of the Sparser system.

"grammar" is a directory containing a number of subdirectories at several levels depending on what grammar (if any) is distributed with your version of Sparser.

"load Sparser" is a short file of Lisp source code used to bootstrap the loading of the system as described next.

2.2.2 Loading

To load the files of the Sparser system into an active Lisp session, you must first have the session running, using whatever procedure is appropriate to the version of Common Lisp that you are using, and then you simply 'load' the file "…:Sparser:load Sparser" and wait the approximately fifteen minutes that it typically takes to complete the process.

This file—"load Sparser"—must be edited the first time that you use Sparser so that the direct call that it makes to the master-loader file within the "f" directory has the correct prefix, reflecting the location at which you have positioned the "Sparser" directory on your disk. Instructions indicating what should be changed are included in the file.

2.2.3 Creating an image

When running on the Macintosh under MCL 2.0 one can save out a image of the loaded Sparser system as a stand-alone Macintosh "application" that incorporates all the aspects of the Lisp system that Sparser needs and can be launched independently from the Finder. If one uses the Sparser routine for creating this application then the resulting application will look for and load certain source files when it is launched. These files are intended to:

- If you have a source license to Sparser, you can arrange to have new files loaded at launch-time that were not included when the image was created or which reflect new versions of already loaded files. This is to facilitate the development of incremental extensions to Sparser's grammar or its interfaces where one will be changing some files and adding new ones and does not want to take the time to load the entire system with each session.

- For any version of Sparser, you can specialize the default run-time environment, possibly changing the values that were established when the image was made to reflect your experience or to make a change that is to have an effect for just one session.

When using a Macintosh MCL version of Sparser the option to create an image is given automatically at the end of the loading sequence. You will be asked if you want to create an image (to which you reply "y" or "n" for yes or no), and if you say yes you will then be asked whether the version is for

"development". A development version will look for the "launch" configuration file when it is launched and load it if it is present; otherwise no files will be accessed, which can be convenient if the application is kept separately from the Sparser directory.

If you say that you want an image, it will be created immediately by using the `save-application` function provided by MCL 2.0. The application will initially have a name based on Sparser's version and the current date, e.g. "Sparser v2.3 2/14", and will be placed as a daughter of the "Sparser" directory. As with any application, you can then change its name and location to whatever you like.

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.