LISP - FILE I/O

We have discussed about how standard input and output is handled by common LISP. All these functions work for reading from and writing into text and binary files too. Only difference is in this case the stream we use is not standard input or output, but a stream created for the specific purpose of writing into or reading from files.

In this chapter we will see how LISP can create, open, close text or binary files for their data storage.

A file represents a sequence of bytes, does not matter if it is a text file or binary file. This chapter will take you through important functions/macros for the file management.

Opening Files

You can use the **open** function to create a new file or to open an existing file. It is the most basic function for opening a file. However, the **with-open-file** is usually more convenient and more commonly used, as we will see later in this section.

When a file is opened, a stream object is constructed to represent it in the LISP environment. All operations on the stream are basically equivalent to operations on the file.

Syntax for the **open** function is:

```
open filename &key :direction :element-type :if-exists :if-does-not-exist
:external-format
```

where,

- The *filename* argument is the name of the file to be opened or created.
- The *keyword* arguments specify the type of stream and error handling ways.
- The **:direction** keyword specifies whether the stream should handle input, output, or both, it takes the following values:
    - :input - for input streams *defaultvalue*
    - :output - for output streams
    - :io - for bidirectional streams
    - :probe - for just checking a files existence; the stream is opened and then closed.
- The **:element-type** specifies the type of the unit of transaction for the stream.
- The **:if-exists** argument specifies the action to be taken if the :direction is :output or :io and a file of the specified name already exists. If the direction is :input or :probe, this argument is ignored. It takes the following values:
    - :error - it signals an error.
    - :new-version - it creates a new file with the same name but larger version number.
    - :rename - it renames the existing file.
    - :rename-and-delete - it renames the existing file and then deletes it.
    - :append - it appends to the existing file.
    - :supersede - it supersedes the existing file.
    - nil - it does not create a file or even a stream just returns nil to indicate failure.
- The **:if-does-not-exist** argument specifies the action to be taken if a file of the specified name does not already exist. It takes the following values:

- :error - it signals an error.
- :create - it creates an empty file with the specified name and then uses it.
- nil - it does not create a file or even a stream, but instead simply returns nil to indicate failure.

- The **:external-format** argument specifies an implementation-recognized scheme for representing characters in files.

For example, you can open a file named myfile.txt stored in the /tmp folder as:

```
(open "/tmp/myfile.txt")
```

Writing to and Reading from Files

The **with-open-file** allows reading or writing into a file, using the stream variable associated with the read/write transaction. Once the job is done, it automatically closes the file. It is extremely convenient to use.

It has the following syntax:

```
with-open-file (stream filename {options}*)
   {declaration}* {form}*
```

- *filename* is the name of the file to be opened; it may be a string, a pathname, or a stream.
- The *options* are same as the keyword arguments to the function open.

Example 1

Create a new source code file named main.lisp and type the following code in it.

```
(with-open-file (stream "/tmp/myfile.txt" :direction :output)
   (format stream "Welcome to Tutorials Point!")
   (terpri stream)
   (format stream "This is a tutorials database")
   (terpri stream)
   (format stream "Submit your Tutorials, White Papers and Articles into our Tutorials
Directory.")
)
```

Please note that all input-output functions discussed in the previous chapter, such as, terpri and format are working for writing into the file we created here.

When you execute the code, it does not return anything; however, our data is written into the file. The **:direction :output** keywords allows us do this.

However, we can read from this file using the **read-line** function.

Example 2

Create a new source code file named main.lisp and type the following code in it.

```
(let ((in (open "/tmp/myfile.txt" :if-does-not-exist nil)))
   (when in
      (loop for line = (read-line in nil)

      while line do (format t "~a~%" line))
      (close in)
   )
)
```

When you execute the code, it returns the following result:

```
Welcome to Tutorials Point!
This is a tutorials database
Submit your Tutorials, White Papers and Articles into our Tutorials Directory.
```

Closing File

The **close** function closes a stream.