Parsetoml Documentation

Release 0.2.0

Maurizio Tomasi

May 18, 2018

Contents

Parsetoml is a Nim library that parses text files written in TOML format.

CHAPTER 1

Introduction

This manual describes Parsetoml, a Nim library to parse TOML files. The library is meant to be compatible with version 0.3.1 of the TOML specification. It implements a streaming parser, i.e., a parser which does not hold the whole file to parse in memory but rather reads one character at a time. The parser outputs a tree data structure based on the type *TomlTableRef*.

In this section we provide a short overview of the usage of the library. We will use the following TOML file as an example:

```
[files]
input_file_name = "test.txt"
output_file_name = "output.txt"

[[filters]]
command = "head"
lines = 5

[[filters]]
command = "cut"
fields = [1,2,4]
```

The purpose of this TOML file is to specify how to apply certain filters to an input text file, and where the result should be saved. It describes the following shell command:

```
cat test.txt | head -n 5 |  cut -f 1,2,4 > output.txt
```

1.1 Parsing a TOML file

To parse a file, there are a few functions that can be used. We'll use the most straightforward one, *parseFile()*: it can either accept a `File` object or a string containing the name of the file to read. Assuming that the name of the TOML file above is `test.toml`, we can therefore read the file in memory and dump its internal representation with the following code:

```
import parsetoml

let data = parsetoml.parseFile("test.toml")
parsetoml.dump(data)
```

(For the sake of clarity, we refer to functions from the Parsetoml library with a full qualification, e.g., `parsetoml.parseFile`. This is however not necessary.) The output of the program is the following:

```
files = table
    input_file_name = string("test.txt")
    output_file_name = string("output.txt")
filters[0] = table
    command = string("head")
    lines = int(5)
filters[1] = table
    command = string("cut")
    fields = array(int(1)int(2)int(4))
```

The purpose of the `dump()` function is to write a readable representation of the tree of nodes created by functions like *`parseFile()`*. It is meant primarly for debugging, and it is a good tool to understand how Parsetoml works internally.

1.2 Navigating through the contents of a TOML file

The `data` variable has type *`TomlTableRef`*, which is a reference to a *`TomlTable`* object, i.e., to an ordered table which associates strings (the keys in the TOML file, e.g., `input_file_name`) with values. The latter are represented by a *`TomlValueRef`* type.

CHAPTER 2

Installation

To install the Parsetoml library, you can use Nimble:

```
nimble install parsetoml
```

The Git repository containing the development version is available on GitHub at the address https://github.com/ziotom78/parsetoml.

CHAPTER 3

Parse functions

In this section we provide a description of the functions in the Parsetoml library that read a textual representation of a TOML file and return a tree of nodes.

3.1 Data types

object**TomlTableRef**
: A reference to a *TomlTable*. This is the default return value for all the functions that parse text in TOML format.

object**TomlTable**
: This data type is used by Parsetoml to associate key names with TOML values. The library uses a `OrderedTable` instead of a `Table`, so that the order of declaration of the keys in the TOML file is preserved.

object**TomlValueRef**
: A reference to a *TomlValue*. Objects of this kind populate *TomlTable* objects.

object**TomlValue**
: The value associated with a key in a TOML file. It is a parametric type defined by the following code:

```
TomlValue* = object
    case kind* : TomlValueKind
    of TomlValueKind.None: nil
    of TomlValueKind.Int: intVal* : int64
    of TomlValueKind.Float: floatVal* : float64
    of TomlValueKind.Bool: boolVal* : bool
    of TomlValueKind.Datetime: dateTimeVal* : TomlDateTime
    of TomlValueKind.String: stringVal* : string
    of TomlValueKind.Array: arrayVal* : seq[TomlValueRef]
    of TomlValueKind.Table: tableVal* : TomlTableRef
```

object**TomlError**
: This exception object is used by Parsetoml to signal errors happened during the parser of text. It has just one

field, `location`, which is of type ParserState and has the following public fields:

Field	Type	Description
`fileName`	`string`	Name of the file being parsed (might be "")
`line`	`int`	Number of the line where the error was detected
`column`	`int`	Number of the column where the error was detected
`stream`	`streams.Stream`	Input stream

The following example shows how to properly signal an error during the parsing of a TOML file to the user:

```
try:
    # Parse some TOML file here
except parsetoml.TomlError:
    # Retrieve information about the location of the error
    let loc = (ref parsetoml.TomlError)(getCurrentException()).location
    # Print a nicely formatted string explaining what went wrong
    echo(loc.fileName & ":" & $(loc.line) & ":" & $(loc.column)
         & ": " & getCurrentExceptionMsg())
```

3.2 Procedures

The Parsetoml library provides several functions to parse text in TOML format. Here is an example of application of the *parseString()* procedure:

```
import parsetoml

# We define a "Parameters" tuple which is initialized using data
# from a TOML file.
type
    Parameters = tuple
        foo : string
        bar : int64

proc parseParams(tree : TomlTableRef) : Parameters =
    result.foo = tree.getString("input.foo")
    result.bar = tree.getInt("input.bar")

let tree = parsetoml.parseString("""
[input]
foo = "a"
bar = 14
""")

let params = parseParams(tree)
assert params.foo == "a"
assert params.bar == 14
```

proc **parseString**(tomlStr : string, fileName : string = "") → *TomlTableRef*

Assuming that *tomlStr* is a string containing text in TOML format, this function parses it and returns a reference to a newly created *TomlTable* object.

Errors in *tomlStr* are signaled by raising exceptions of type *TomlError*. The `location.fileName` field of the exception itself will be set to *fileName*.

proc**parseStream**(inputStream : streams.Stream, fileName : string = "") → *TomlTableRef*

This function is similar to `parseString()`, but it reads data from *inputStream*. The stream is parsed while it is being read (i.e., the parsing does not have to wait till the whole file has been read in memory).

proc**parseFile**(f : File, fileName : string = "") → *TomlTableRef*

The same as `parseStream()`, but this procedure accepts a `File` instead of a `streams.Stream`.

proc**parseFile**(fileName : string) → *TomlTableRef*

This is a wrapper to the previous implementation of `parseFile`: it handles the opening/closing of the file named *fileName* automatically.

CHAPTER 4

Accessing keys and values in a parsed TOML tree

Once a TOML file has been parsed, the data are available in a *TomlTableRef* object, which implements a tree-like structure. The Parsetoml library provides several ways to access the information encoded in the tree:

1. Generic functions (easy)
2. Tree traversal (complex but powerful)

4.1 TOML addresses

An *address* is a string which identifies the position of a key/value pair within a TOML file. Parsetoml allows to quickly retrieve the value of a key given its address, without the need of traversing the whole TOML tree.

Addresses can reference sub-tables as well as elements of table arrays: the names of nested sub-tables are separated by dots, while integer numbers within square brackets indicate elements of a table array.

As an instance, consider the following TOML table:

```
[[fruit]]
  name = "apple"

  [fruit.physical]
    color = "red"
    shape = "round"

  [[fruit.variety]]
    name = "red delicious"

  [[fruit.variety]]
    name = "granny smith"

[[fruit]]
  name = "banana"
```

(continues on next page)

(continued from previous page)

```
  [[fruit.variety]]
    name = "plantain"
```

This TOML file contains an array of tables named `fruit`; the array contains two elements: an `apple` and a `banana`. Each element contains an additional array of tables, named in both cases `fruit.variety`: the apple element has two varieties, while the banana element has just one.

The value `granny smith` is associated to the key whose address is `fruit[0].variety[1].name`, while the value `round` is associated to the key `fruit[0].physical.shape`. Here is a commented version of the TOML file where each key/value pair has its address spelled explicitly:

```
[[fruit]]
  name = "apple"              # fruit[0].name

  [fruit.physical]
    color = "red"             # fruit[0].physical.color
    shape = "round"           # fruit[0].physical.color

  [[fruit.variety]]
    name = "red delicious"    # fruit[0].variety[0].name

  [[fruit.variety]]
    name = "granny smith"     # fruit[0].variety[1].name

[[fruit]]
  name = "banana"             # fruit[1].name

  [[fruit.variety]]
    name = "plantain"         # fruit[1].variety[0].name
```

4.2 Generic functions

Each of the functions listed in this section returns the value associated with the key whose address is passed in the parameters named either *fullAddr* or *address*.

proc**`getValueFromFullAddr`**(table : TomlTableRef, fullAddr : string) → *TomlValueRef*

Looks for an element in *table* that matches the address *fullAddr* and return the corresponding value. If no match is found, a *`TomlValueRef`* object with kind `TomlValueKind` equal to `None` is returned.

proc**`getInt`**(table : TomlTableRef, address : string) → int64

Wrapper to :nim:proc::*getValueFromFullAddr*. if *address* points to an key that does not exists, or if the type of the key pointed by *address* is not an integer, a `KeyError` exception is raised.

proc**`getInt`**(table : TomlTableRef, address : string, default : int64) → int64

Wrapper to :nim:proc::*getValueFromFullAddr*. if the type of the key pointed by *address* is not an integer, a `KeyError` exception is raised. However, if the key does not exist, the value of *default* will be returned instead (i.e., no `KeyError` exception is raised).

proc**`getFloat`**(table : TomlTableRef, address : string) → float64

Wrapper to :nim:proc::*getValueFromFullAddr*. if *address* points to an key that does not exists, or if the type of the key pointed by *address* is not a floating point value, a `KeyError` exception is raised.

proc**`getFloat`**(table : TomlTableRef, address : string, default : float64) → float64

Wrapper to :nim:proc::*getValueFromFullAddr*. if the type of the key pointed by *address* is not a floating point

value, a `KeyError` exception is raised. However, if the key does not exist, the value of *default* will be returned instead (i.e., no `KeyError` exception is raised).

proc**`getBool`**(table : TomlTableRef, address : string) → bool
Wrapper to :nim:proc::*getValueFromFullAddr*. if *address* points to an key that does not exists, or if the type of the key pointed by *address* is not a Boolean, a `KeyError` exception is raised.

proc**`getBool`**(table : TomlTableRef, address : string, default : bool) → bool
Wrapper to :nim:proc::*getValueFromFullAddr*. if the type of the key pointed by *address* is not a Boolean, a `KeyError` exception is raised. However, if the key does not exist, the value of *default* will be returned instead (i.e., no `KeyError` exception is raised).

proc**`getString`**(table : TomlTableRef, address : string) → string
Wrapper to :nim:proc::*getValueFromFullAddr*. if *address* points to an key that does not exists, or if the type of the key pointed by *address* is not a string, a `KeyError` exception is raised.

proc**`getString`**(table : TomlTableRef, address : string, default : string) → string
Wrapper to :nim:proc::*getValueFromFullAddr*. if the type of the key pointed by *address* is not a string, a `KeyError` exception is raised. However, if the key does not exist, the value of *default* will be returned instead (i.e., no `KeyError` exception is raised).

proc**`getDateTime`**(table : TomlTableRef, address : string) → parsetoml.TomlDateTime
Wrapper to :nim:proc::*getValueFromFullAddr*. if *address* points to an key that does not exists, or if the type of the key pointed by *address* is not a date/time, a `KeyError` exception is raised.

proc**`getDateTime`**(table : TomlTableRef, address : string, default : parsetoml.TomlDateTime) → parsetoml.TomlDateTime
Wrapper to :nim:proc::*getValueFromFullAddr*. if the type of the key pointed by *address* is not a date/time, a `KeyError` exception is raised. However, if the key does not exist, the value of *default* will be returned instead (i.e., no `KeyError` exception is raised).

4.3 Tree traversal

It is possible to directly access the fields of a *`TomlTableRef`* to perform a tree traversal of the data structure. The implementation of the `dump()` procedure is extremely interesting in this respect:

```
proc dump*(table : TomlTableRef, indentLevel : int = 0) =
    let space = spaces(indentLevel)
    for key, val in pairs(table):
        if val.kind == TomlValueKind.Table:
            echo space & key & " = table"
            dump(val.tableVal, indentLevel + 4)
        elif (val.kind == TomlValueKind.Array and
              val.arrayVal[0].kind == TomlValueKind.Table):
            for idx, val in val.arrayVal:
                echo space & key & "[" & $idx & "] = table"
                dump(val.tableVal, indentLevel + 4)
        else:
            echo space & key & " = " & $(val[])
```

A good source of information is the source code itself, which can be found on the GitHub website.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

G

P