PyOrgMode

14 December 2010

Contents

1 PyOrgMode **1**
1.1 Tools . . . 1
1.2 Documentation . . . 1
1.2.1 **TODO** TODO LIST [0%] . . . 1
1.2.2 BUG LIST [0%] . . . 2
1.2.3 ChangeLog . . . 2
1.2.4 Authors [1/1] . . . 2
1.3 Code . . . 2
1.3.1 License . . . 2
1.3.2 Imports . . . 2
1.3.3 Class OrgElement . . . 2
1.3.4 Class Property . . . 2
1.3.5 Class Schedule . . . 2
1.3.6 Class Drawer . . . 2
1.3.7 Class Node . . . 2
1.3.8 Class DataStructure . . . 2

1 PyOrgMode

1.1 Tools

`Tangle` (Export the PyOrgMode.py file)

1.2 Documentation

1.2.1 TODO TODO LIST [0%]

- ☐ Error/Warning managment
- ☐ TODO Document every function correctly (docstrings)
- ☐ TODO Check for other OS compatibility
- ☐ TODO Do a validator (input file MUST be output file, and check every function)
- ☐ TODO TODO tags (and others)
- ☐ TODO Priority
- ☐ TODO Add more types of data (List,...)
- ☐ TODO Class DataStructure : Modular way to check for elements (the finding loop can became rather big if everything is implemented)
- ☐ TODO Class DataStructure : Move or delete nodes

1.2.2 BUG LIST [0%]

☐ The drawers lost indentation and added spaces/tabs in properties :NON-BLOCKING::NODATALOSS:

1.2.3 ChangeLog

- 0.01f
  - New elements
    - Added Schedule element for ‘`DEADLINE:` and ‘`SCHEDULED:`

1.2.4 Authors [1/1]

- ☒ BISSON Jonathan <bissonjonathan on the googlethingy>

1.3 Code

1.3.1 License

1.3.2 Imports

1.3.3 Class OrgElement

1.3.4 Class Property

1.3.5 Class Schedule

1.3.6 Class Drawer

1.3.7 Class Node

1.3.8 Class DataStructure

```
class DataStructure:
    """
    Data structure containing all the nodes
    The root property contains a reference to the level 0 node
    """

    root = None

    def append(self,node):
        if node.parent is None: # Node has no parent (so it is the level 0 node)
            self.root = node # So parent is the first added node
        else:
            node.parent.append(node)

    def load_from_file(self,name):
        current = Node()
        parent = None
        file = open(name,'r')

        re_heading_stars = re.compile("^(\*+)\s(.*?)\s*$")
        re_drawer = re.compile("^(?:\s*?)(?::)(\S.*?)(?::)\s*(.*?)$")
        re_heading = re.compile("(?:\*+)\s((.*?)(?:\s*.*?)\s*\s)((:\S(.*?)\S:$)|$)")

        # The (?!.*?\]) protects against links containing tags being considered as tag
        re_tags = re.compile("(?:::|\s:)(\S.*?\S)(?=:)(?!.*?\])")
```

```
re_scheduled = re.compile("(?:\s*)(SCHEDULED|DEADLINE)(?::\s*)(<.*?>)(?:\s.*|$

current_drawer = None
for line in file:
    heading_stars = re_heading_stars.search(line)
    drawer = re_drawer.search(line)
    scheduled = re_scheduled.findall(line)

    if isinstance(current, Drawer):
        if drawer:
            if drawer.group(1) == "END":
                current = current.parent
            elif drawer.group(2):
                current.append(Property(drawer.group(1),drawer.group(2)))
        else:
            current.append(line.rstrip("\n"))
    elif drawer:
        current = current.append(Drawer(drawer.group(1)))

    elif heading_stars: # We have a heading
        self.append(current) # We append the current node as it is done

        # Is that a new level ?
        if (len(heading_stars.group(1)) > current.level): # Yes
            parent = current # Parent is now the current node

        # If we are going back one or more levels, walk through parents
        while len(heading_stars.group(1)) < current.level:
            current = current.parent

        # Creating a new node and assigning parameters
        current = Node()
        current.level = len(heading_stars.group(1))
        current.heading = re_heading.findall(line)[0][0].rstrip("\n")
        current.parent = parent

        # Looking for tags
        current.tags = re_tags.findall(line)
    elif \texttt{scheduled:}
```

```
                current.append(Schedule(scheduled[0][0], scheduled[0][1]))
            else: # Nothing special, just content
                if line is not None:
                    current.append(line)

        # Add the last node
        if current.level>0:
            self.append(current)

        file.close()

    def save_to_file(self,name):
        output = open(name,'w')
        output.write(str(self.root))
        output.close()
```