Ralph-Johan Back | Viorel Preoteasa

A Python Specification of the Tkinter Text-widget

Turku Centre *for* Computer Science

TUCS Technical Report
No 607, January 2005

A Python Specification of the Tkinter Text-widget

Ralph-Johan Back
Viorel Preoteasa
Turku Centre for Computer Science
DataCity, Lemminkäisenkatu 14 A
FIN-20520 Turku, Finland.

TUCS Technical Report
No 607, January 2005

Abstract

Writing programs means in addition of using a programming language to use libraries of objects or functions. Looking at the documentation of Tkinter Text-widget and trying to use it we have encountered the problem of unspecified consistent behavior of it. In this paper we give an alternative of specifying and documenting a piece of code giving a functional python module that is intended to behave as the original Tkinter Text-widget.

1 Introduction

The Tkinter Text-widget is the basis for some graphical user interface applications developed in our department and written in Python. Hence, it is important to understand how it works.

Trying to use the Text-widget documentation leads to the problem of unspecified consistent behavior. For example if a piece of text from the text-widget has set a tag and a string is inserted in the middle of it then the inserted text will be marked with this tag. Moreover, when a string is inserted in the Text-widget it will be marked with the tags that are common for the characters that are at the left and right of the inserted string. This is a useful feature that is not specified in the Tkinter documentation [3]. Even the book [2] does not mention this feature and many others.

Here we give a precise definition of the Text-widget, in terms of a collection of classes that have been constructed for the purpose of specification. These classes have no a priori relationship with the actual classes that are used to implement the widget.

This approach is useful also for reasoning about a program that uses the Text-widget and for proving properties of it.

We used the spewise feature introduction [1], a technique to built object oriented software systems by adding only one feature at a time. The system should be built in layers where any new layer introduces a new feature such that the functionality of the previous layers is preserved. Each layer should be a running system that has all features introduced by it and the previous layers.

2 The Classes Diagram of the Tkinter Text widget

Description

The description of the Tkinter Text-widget uses a number of techniques. The basis class is TextStrings, which has a very simple notion of a text, essentially considering it to be a string. This simple notion is generalized in the class TextWithElements, so that the string elements can be any elements that have a public field called char. Thus, the new class Element is just a wrapper around the class/type Char. This is thus an example of delegation (or actually just forwarding).

Inheritance is used to add one feature at a time to the basic widget.In this way, TextWithElements inherits the interface from the TextStrings class, but generalizes the methods to work with any kind of elements. The class TextWithMarks adds marks to the text, while the class TextWithTags adds tags, by extending the Element class to a ElementsWithTags class. These are both independent extensions of the class TextStrings.

The class TextWithTagConfig adds information about the tag bindings (the values of options for the tag).

The class TextWithMarksTags now combines the features of TextWithMarks and TextWithTags, using multiple inheritance.

Finally, the class Text is a wrapper around the class TextWithMarksTags, which uses indexes rather than position numbers to identify positions in the text. This is thus again an example of delegation. This class uses an auxilliary class Index, that allows references to positions in text by (line,column) rather than by just position.

The specification is split into this large number of classes, in order to clearly show how each new mechanism is introduced into the specification. Each class only introduces one mechanism, which is clearly distinguished from other mechanisms, and which does not invalidate previous mechanisms. The basic methods used are, as explained above, inheritance and forwarding/delegation. The class Index is a slight abstraction of the way the indexes are represented in Tkinter: each index is really represented as a text string of the form "line.column", from which the line and column values can be extracted. A further class called StringIndex could be introduced for such strings.

The most problematic part of the specification is how to deal refer to positions in text. There seems to be essentially two possible ways,

1. we consider a text to be essentially a sequence of elements, and refer to positions by sequence elements, or

2. we consider a text to be a sequence of lines, where each line is a sequence of elements, and refer to positions directly by (line, column) indexes.

Here we have chosen the first alternative, because it seems more intuitive, and simplifies the definition of insert, delete and get operations on the text. On the other hand, it makes the handling of indexes more complicated. The latter alternative might be equally good, and would considerably simplify the treatment of indexes, at the cost of making insert, delete and get operations more complicated (as well as all complicated other operations that require scanning the whole text). The alternative definition could also be described here, in oder to compare these two specifications.

Comments

We have tested this definition by executing it as a Python module. This execution have provided a syntactic check of the specification, as well as testing that there are no obvious errors in the specification.

Structure

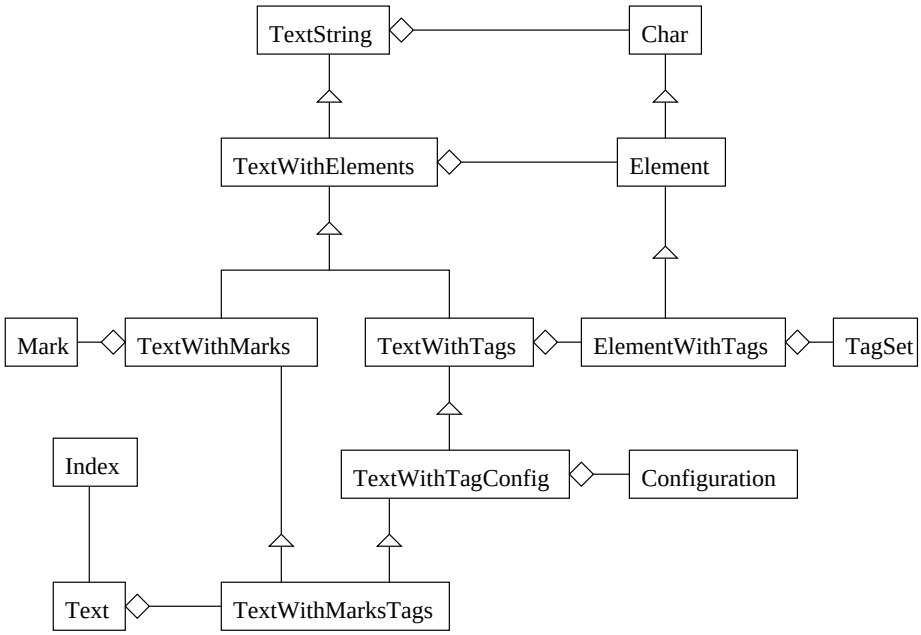


Figure 1

3 TextString

Purpose

This class introduces the basic class for manipulating texts, seen as simply strings of characters. It has methods for inserting, deleting and getting the text.

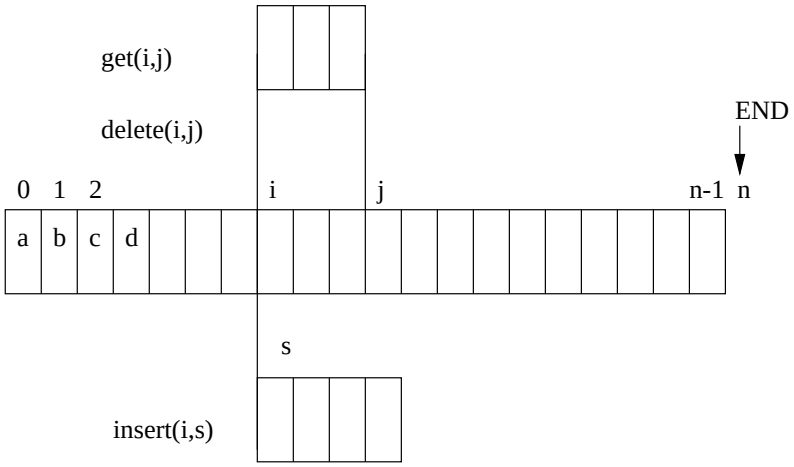
Structure

TextString
-str
+__init__() +__invariant__() +END() +positionOK(i) +insert(i, s) +delete(i, j) +get(i, j)

```
class TextString:
    def __init__(self):
        self.str := ''
    def __invariant__(self):
        true
```

```
def END(self):
    return len(self.str)
def positionOK(self, i):
    return 0 ≤ i ≤ self.END()
def insert(self, i, s):
    assert self.positionOK(i)
    self.str := self.str[:i] + s + self.str[i:]
def delete(self, i, j := None):
    if j=None:
        j := i + 1
    assert self.positionOK(i) and self.positionOK(j) and (i ≤ j)
    self.str := self.str[:i] + self.str[j:]
def get(self, i, j:=None):
    if j=None:
        j := i + 1
    assert self.positionOK(i) and self.positionOK(j) and (i ≤ j)
    return self.str[i:j]
def __getitem__(self,i):
    assert 0 ≤ i < len(self.str)
    return self.str[i]
```

Description



Special cases

We analyze here a collection of special cases, to see that the definitions of the methods correspond to our intuition. Essentially, our goal is to simplify calls with special parameters by using basic properties of sequences, to see that the properties that we get conform to our intuition.

Assume below that self is as in picture, with elements numbered $0, \ldots, n-1$. Assume that parameters satisfy method preconditions.

insert

```
self.insert(0,s):

    self.insert(0,s)
```

= {parameter substitution}

self.str := self.str[:0] + s + self.str[0:]

$=$ {str[:0] = '', str[0:] = str}

self.str := '' + s + self.str

$=$ {'' + s = s}

self.str := s + self.str

self.insert(self.END(),s):

self.insert(self.END(),s)

$=$ {parameter substitution}

self.str := self.str[:self.END()] + s + self.str[self.END():]

$=$ {self.END() = n}

self.str := self.str[:n]+s+self.str[n:]

$=$ {self.str[:n] = self.str, self.str[n:] = ''}

self.str := self.str + s + ''

$=$ {s + '' = s}

self.str := self.str + s

self.insert(i, ''):

self.insert(i, '')

$=$ {parameter substitution}

self.str := self.str[:i] + '' + self.str[i:]

$=$ {''+s = s}

self.str := self.str[:i] + self.str[i:]

$=$ {str[:i] + str[i:] = str}

self.str := self.str

$=$ {identity}

pass

delete

self.delete(0,j):

self.delete(0,j)

$=$ {parameter substitution}

self.str := self.str[:0] + self.str[j:]

$=$ {self.str[:0] = ''}

self.str := '' + self.str[j:]

$=$ { '' + s = s}

self.str := self.str[j:]

self.delete(i, self.END()):

self.delete(i, self.END())

$=$ {parameter substitution}

self.str := self.str[:i] + self.str[self.END():]

$=$ {self.str[self.END():] = ''}
self.str =self.str[:i] + ''
$=$ {s + '' = s}
self.str := self.str[:i]

self.delete(0, self.END()):

self.delete(0, self.END())
$=$ {parameter substitution}
self.str = self.str[:0] + self.str[self.END():]
$=$ {s[:0] = '', s[len(s):] = ''}
self.str := '' + ''
$=$
self.str := ''

self.delete(i, i):

self.delete(i, i)
$=$ {parameter substitution}
self.str := self.str[:i] + self.str[i:]
$=$ {s[:i] + s[i:] = s}
self.str := self.str
$=$
pass

self.delete(0):

self.delete(0)
$=$ {parameter substitution}
self.str := self.str[:0] + self.str[0+1:]
$=$ {s[:0] = ''}
self.str := self.str[1:]

Properties

s := self.get(i,j)
self.delete(i,j)
self.insert(i,s)

$=$ {definitions}

s := self.str[i:j]
self.str := self.str[:i] + self.str[j:]
self.str := self.str[:i] + s + self.str[i:]

$=$

self.str := (self.str[:i] + self.str[j:])[:i] + self.str[i:j] + (self.str[:i] + self.str[j:])[i:]

$=$

self.str := self.str[:i] + self.str[i:j] + self.str[j:]

=

self.str := self.str

=

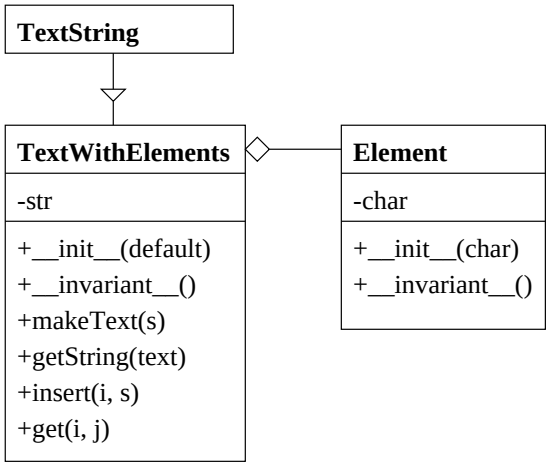
pass

4 TextWithElements

Purpose

This class generalizes the basic TextString class, by considering each element to be an instance of a class Element, rather than just a character.

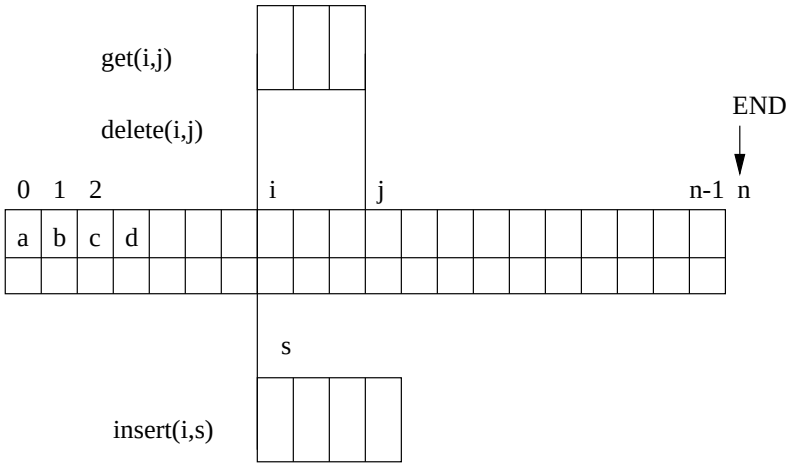
Structure




```
class Element:
    def __init__(self, char):
        self.char := char
    def __invariant__(self):
        IsChar(self.char)
class TextWithElements(TextString):
    def __init__(self, default):
        self.str := [ ]
        self.default := default
    def __invariant__(self):
        for elem in self.str:
            elem.__invariant__()
    def makeText(self, s):
        text := [ ]
        for c in s:
            elem := self.default(c)
            text.append(elem)
        return text
    def getString(self, text):
        s := ''
        for elem in text:
            s := s + elem.char
```

```
        return s
    def insert(self, i, s):
        assert self.positionOK(i) and IsString(s)
        text := self.makeText(s)
        self.str := self.str[:i] + text + self.str[i:]
    def get(self, i, j:=None):
        return self.getString(TextString.get(self, i, j))
    def __getitem__(self,i):
        return TextString.__getitem__(self,i).char
```

Description



Special cases

Assume that self is as in picture, with elements in $0, \ldots, n-1$.

Assume that parameters satify preconditions

insert

```
self.insert(i,'ab'):

    self.insert(i,'ab')

=

    text := self.makeText(s)
    self.str := self.str[:i]+text+self.str[i:]

=

    text := [ ]
    for c in 'ab':
        elem := self.default(c)
        text.append(elem)
    self.str := self.str[:i] + text + self.str[i:]

=

    text := [ ]
    c := 'a'
    elem := self.default(c)
    text.append(elem)
    c := 'b'
```

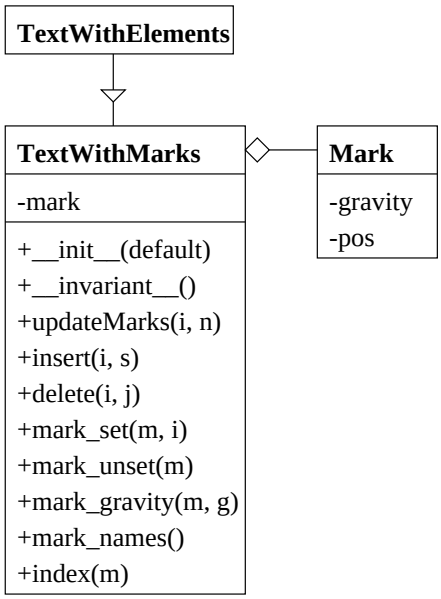
```
        elem := self.default(c)
        text.append(elem)
        self.str := self.str[:i] + text + self.str[i:]
    =
        self.str[:i] + [self.default('a'), self.default('b')] + self.str[i:]
```

5 TextWithMarks

Purpose

This class introduces a mechanism for marking specific positions in the text. Two standard marks, CURRENT (present cursor position) and INSERT (text cursor position) are predefined.

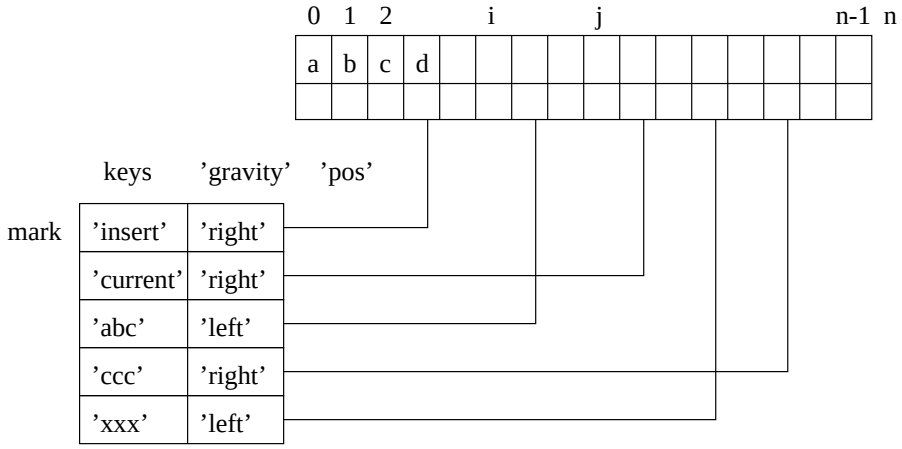
Structure




```
INSERT   := 'insert'
CURRENT := 'current'
RIGHT    := 'right'
LEFT     := 'left'
class TextWithMarks(TextWithElements):
    def __init__(self, default):
        TextWithElements.__init__(self, default)
        self.mark := {}
        self.mark[INSERT] := {'pos':0, 'gravity':RIGHT}
        self.mark[CURRENT] := {'pos':0, 'gravity':RIGHT}
    def __invariant__(self):
        TextWithElements.__invariant__(self)
        for m in self.mark.keys():
            self.positionOK(self.mark[m]['pos'])        # (1)
            self.mark[m]['gravity'] in [LEFT, RIGHT]   # (2)
        INSERT in self.mark.keys() and CURRENT in self.mark.keys() # (3)
    def UpdateMarks(self, i, n):
```

```
    for m in self.mark.keys():
        if self.mark[m]['pos'] > i:
            self.mark[m]['pos'] := self.mark[m]['pos'] + n
        elif (self.mark[m]['pos'] = i) and (self.mark[m]['gravity'] = RIGHT):
            self.mark[m]['pos'] := self.mark[m]['pos'] + n
def insert(self, i, s):
    TextWithElements.insert(self, i, s)
    self.UpdateMarks(i, len(s))
def delete(self, i, j:=None):
    if j=None:
        j := i + 1
    TextWithElements.delete(self, i, j)
    for m in self.mark.keys():
        if self.mark[m]['pos'] in range(i, j):
            self.mark[m]['pos'] := i
        elif self.mark[m]['pos'] ≥ j:
            self.mark[m]['pos'] := self.mark[m]['pos'] - (j-i)
def mark_set(self, m, i):
    assert self.positionOK(i)
    if not m in self.mark.keys():
        self.mark[m] := {'pos':i, 'gravity':RIGHT}
    else:
        self.mark[m]['pos'] := i
def mark_unset(self, m):
    assert m in self.mark.keys() and m≠INSERT and m≠CURRENT
    del self.mark[m]
def mark_gravity(self, m , g:=None):
    assert m in self.mark.keys()
    if g = None:
        return self.mark[m]['gravity']
    else:
        assert g in [LEFT, RIGHT]
        self.mark[m]['gravity'] := g
def mark_names(self):
    return self.mark.keys()
def index(self, m):
    assert m in self.mark.keys()
    return self.mark[m]['pos']
```

Description



Preserving invariant

It is rather easy to prove that all functions preserve invariants. It is obvious that __init__ establishes the invariant and mark_set, mark_unset, mark_gravity, mark_names preserve the invariant. We prove that delete preserve the invariant. Let x be an instance of TextWithMarks, let n be len(x.str) and i, j be such that $0 \leq$ i, j$\leq n$ and i$\leq$j. Then we should prove:

{x.__invariant__()}
x.delete(i, j)
{x.__invariant__()}

But the function delete only change x.str and the positions of all marks. So we only need to show that delete preserves (1) from __invariant__.

{$\forall$ m $\in$ x.mark.keys • $0 \leq$ x.mark[m]['pos'] $\leq n$}
x.delete(i, j)
{$\forall$ m $\in$ x.mark.keys • $0 \leq$ x.mark[m]['pos'] $\leq$ len(x.str)}

$\Leftrightarrow$

{$\forall$ m $\in$ x.mark.keys • $0 \leq$ x.mark[m]['pos'] $\leq n$}
x.str := x.str[:i] + x.str[j:]
for m in x.mark.keys
    i$\leq$x.mark[m]['pos']<j $\Rightarrow$ x.mark[m]['pos'] := i
    j$\leq$x.mark[m]['pos'] $\Rightarrow$ x.mark[m]['pos'] := x.mark[m]['pos'] - (j - i)
{$\forall$ m $\in$ x.mark.keys • $0 \leq$ x.mark[m]['pos'] $\leq$ len(x.str)}

$\Leftrightarrow$

{$\forall$ m $\in$ x.mark.keys • $0 \leq$ x.mark[m]['pos'] $\leq n$}
x.str := x.str[:i] + x.str[j:]
{$\forall$ m $\in$ x.mark.keys •
    (x.mark[m]['pos']<i $\Rightarrow$ $0 \leq$ x.mark[m]['pos'] $\leq$ len(x.str)) $\wedge$
    (i$\leq$x.mark[m]['pos']<j $\Rightarrow$ $0 \leq$ i $\leq$ len(x.str)) $\wedge$
    (j$\leq$x.mark[m]['pos'] $\Rightarrow$ $0 \leq$ x.mark[m]['pos'] - (j - i) $\leq$ len(x.str))
}

$\Leftrightarrow$

$\forall$ m $\in$ x.mark.keys $\bullet$ $0 \leq$ x.mark[m]['pos'] $\leq n$
$\Rightarrow$
$\forall$ m $\in$ x.mark.keys $\bullet$
(x.mark[m]['pos']<i $\Rightarrow 0 \leq$ x.mark[m]['pos'] $\leq n$ - (j - i)) $\wedge$
(i$\leq$x.mark[m]['pos']<j $\Rightarrow 0 \leq$ i $\leq n$ - (j - i)) $\wedge$
(j$\leq$x.mark[m]['pos'] $\Rightarrow 0 \leq$ x.mark[m]['pos'] - (j - i) $\leq n$ - (j - i))

$\Leftarrow$ $\{(\forall x \bullet \alpha(x) \Rightarrow \beta(x)) \Rightarrow ((\forall x \bullet \alpha(x)) \Rightarrow (\forall x \bullet \beta(x))\}$

$\forall$ m $\in$ x.mark.keys $\bullet$
$0 \leq$ x.mark[m]['pos'] $\leq n$
$\Rightarrow$
(x.mark[m]['pos']<i $\Rightarrow 0 \leq$ x.mark[m]['pos'] $\leq n$ - (j - i)) $\wedge$
(i$\leq$x.mark[m]['pos']<j $\Rightarrow 0 \leq$ i $\leq n$ - (j - i)) $\wedge$
(j$\leq$x.mark[m]['pos'] $\Rightarrow 0 \leq$ x.mark[m]['pos'] - (j - i) $\leq n$ - (j - i))

$\Leftrightarrow$ $\{$notation p(m)=x.mark[m]['pos']$\}$

$\forall$ m $\in$ x.mark.keys $\bullet$
$0 \leq$ p(m) $\leq n$
$\Rightarrow$
(p(m)<i $\Rightarrow 0 \leq$ p(m) $\leq n$ - (j - i)) $\wedge$
(i$\leq$p(m)<j $\Rightarrow 0 \leq$ i $\leq n$ - (j - i)) $\wedge$
(j$\leq$p(m) $\Rightarrow 0 \leq$ p(m) - (j - i) $\leq n$ - (j - i))

$\Leftrightarrow$ $\{(\alpha \Rightarrow \beta \wedge \gamma) \Leftrightarrow ((\alpha \Rightarrow \beta) \wedge (\alpha \Rightarrow \gamma))$ and
$(\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Leftrightarrow (\alpha \wedge \beta \Rightarrow \gamma)\}$

$\forall$ m $\in$ x.mark.keys $\bullet$
$(0 \leq$ p(m) $\leq n \wedge$ p(m)<i $\Rightarrow 0 \leq$ p(m) $\leq n$ - (j - i)) $\wedge$
$(0 \leq$ p(m) $\leq n \wedge$ i$\leq$p(m)<j $\Rightarrow 0 \leq$ i $\leq n$ - (j - i)) $\wedge$
$(0 \leq$ p(m) $\leq n \wedge$ j$\leq$p(m) $\Rightarrow 0 \leq$ p(m) - (j - i) $\leq n$ - (j - i))

- $0 \leq$ p(m) $\leq n \wedge$ p(m)<i $\Rightarrow 0 \leq$ p(m) $\leq n$ - (j - i)

$\Leftarrow$

- $0 \leq$ p(m)
  $\Leftrightarrow \{0 \leq$ p(m)$\}$
  true
- p(m) $\leq n$ - (j - i)
  $\Leftarrow \{$p(m)<i and transitivity of $\leq\}$
  i $\leq n$ - (j - i)
  $\Leftrightarrow \{$j $\leq n\}$
  true

- $0 \leq$ p(m) $\leq n \wedge$ i$\leq$p(m)<j $\Rightarrow 0 \leq$ i $\leq n$ - (j - i)

$\Leftarrow$

- $0 \leq$ i
  $\Leftrightarrow \{$hypothesys$\}$
  true
- i $\leq n$ - (j - i)
  $\Leftrightarrow \{$j $\leq n\}$
  true

- $0 \leq \mathsf{p(m)} \leq n \wedge \mathsf{j} \leq \mathsf{p(m)} \Rightarrow 0 \leq \mathsf{p(m)} - (\mathsf{j} - \mathsf{i}) \leq n - (\mathsf{j} - \mathsf{i})$

$\Leftrightarrow$

$0 \leq \mathsf{p(m)} \leq n \wedge \mathsf{j} \leq \mathsf{p(m)} \Rightarrow (\mathsf{j} - \mathsf{i}) \leq \mathsf{p(m)} \leq n$

$\Leftarrow$

- $(\mathsf{j} - \mathsf{i}) \leq \mathsf{p(m)}$
  $\Leftarrow$ {transitivity and $\mathsf{j} \leq \mathsf{p(m)}$}
  $\mathsf{j} - \mathsf{i} \leq \mathsf{j}$
  $\Leftrightarrow$ {$0 \leq \mathsf{i}$}
  true
- $\mathsf{p(m)} \leq n$
  $\Leftrightarrow$ {hypothesys}
  true

$\Leftrightarrow$

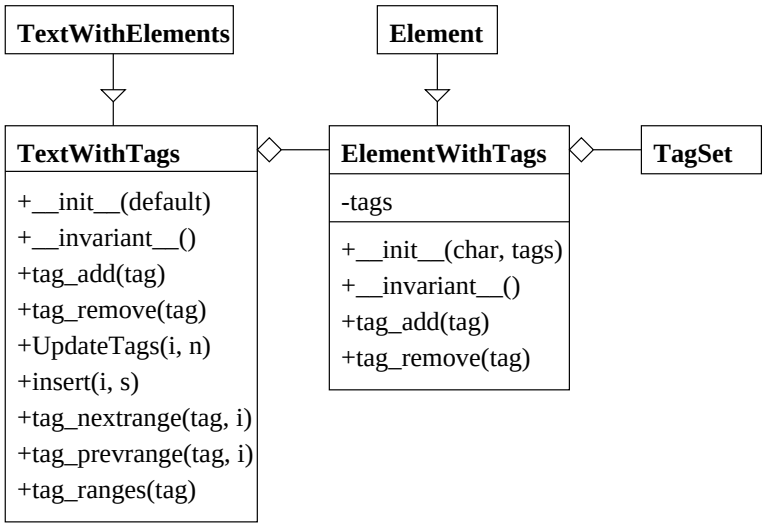
true

6 TextWithTags

Purpose

This class introduces a mechanism that allows the individual elements of the text to be tagged, so that they can be visually and functionally distinguished later on. A standard tag, SEL, is defined for indicating which elements are selected.

Structure




```
class ElementWithTags(Element):
    def __init__(self, char, tags := None):
        Element.__init__(self, char)
        self.tags := {}
        if(tags):
            self.tags.update(tags)
    def __invariant__(self):
        Element.__invariant__(self)
```

```
    def tag_add(self, tag):
        assert IsString(tag)
        self.tags[tag] := None
    def tag_remove(self, tag):
        assert IsString(tag)
        if tag in self.tags.keys():
            del self.tags[tag]

class TextWithTags(TextWithElements):
    def __init__(self, default):
        TextWithElements.__init__(self, default)
    def __invariant__(self):
        TextWithElements.__invariant__(self)
    def tag_add(self, tag, i, j:=None):
        if j = None:
            j := i + 1
        assert self.positionOK(i) and self.positionOK(j) and (i ≤ j) and IsString(tag)
        for k in range(i,j):
            self.str[k].tag_add(tag)
    def tag_remove(self, tag, i, j:=None):
        if j = None:
            j := i + 1
        assert self.positionOK(i) and self.positionOK(j) and (i ≤ j) and IsString(tag)
        for k in range(i,j):
            self.str[k].tag_remove(tag)
    def UpdateTags(self, i, n):
        tags := {}
        if 0 < i and i+n < self.END():
            for t in self.str[i-1].tags.keys():
                if t in self.str[i+n].tags.keys():
                    tags[t] := None
            for elem in self.str[i:i+n]:
                elem.tags.update(tags)
    def insert(self, i, s):
        TextWithElements.insert(self, i, s)
        self.UpdateTags(i, len(s))
    def tag_nextrange(self, tag, i):
        pass
    def tag_prevrange(self, tag, i):
        pass
    def tag_ranges(self, tag):
        assert IsString(tag)
        x := [ ]
        n := 0
        for i in range(self.END()):
            if tag in self.str[i].tags.keys():
                if x:
                    if x[n] = i:
                        x[n] := i + 1
                    else:
                        x[n+1:] := [i, i+1]
                        n := n + 2
                else:
```

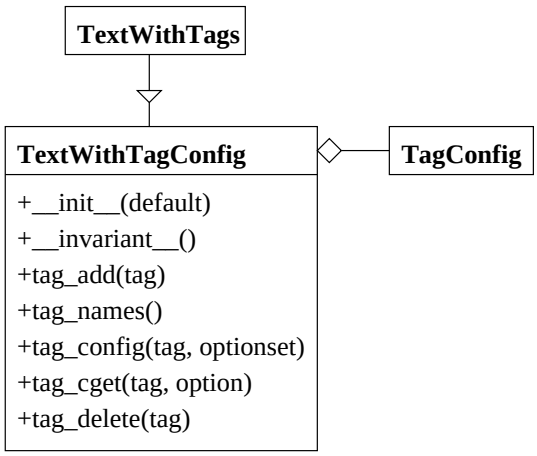
```
                x := [i,i+1]
                n := 1
    return x
```

7 TextWithTagConfiguration

Purpose

This class adds a mechanism for associating with each tag a list of properties (called options). Example properties are colour of character, whether the character is underlined ect.

Structure




```
class TextWithTagConfiguration(TextWithTags):
    def __init__(self, default):
        TextWithTags.__init__(self, default)
        self.tagconfig := {'sel':{}}
    def __invariant__(self):
        TextWithTags.__invariant__(self)
    def tag_add(self, tag, i, j:=None):
        TextWithTags.tag_add(self, tag, i, j)
        if not tag in self.tagconfig.keys():
            self.tagconfig[tag] := {}
    def tag_names(self):
        return self.tagconfig.keys()
    def tag_config(self, tag, optionset):
        assert IsString(tag)
        self.tagconfig[tag] := optionset
    def tag_cget(self, tag, option):
        assert IsString(tag)
        return self.tagconfig[tag][option]
    def tag_delete(self, tag, *tags):
        for t in (tag,) + tags:
            assert IsString(t)
            self.tag_remove(t, 0, self.END())
            if t ≠ 'sel' and t in self.tagconfig.keys():
```

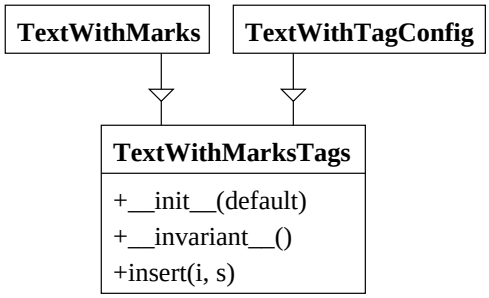
```
        del self.tagconfig[t]
```

8 TextWithMarksTags

Purpose

This class combines the two independent menchanisms that have been described above, text marks and text tags, using multiple inheritance. It should redefine the insert method to update both marks and tags.

Structure



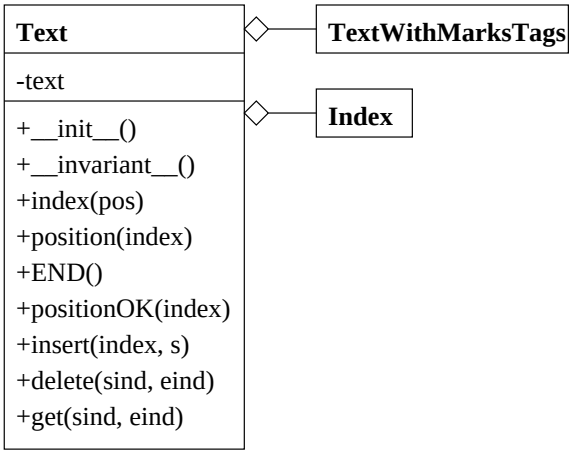

```
class TextWithMarksTags(TextWithMarks, TextWithTagConfiguration):
    def __init__(self, default):
        TextWithMarks.__init__(self, default)
        TextWithTagConfiguration.__init__(self, default)
    def __invariant__(self):
        TextWithMarks.__invariant__(self)
        TextWithTagConfiguration.__invariant__(self)
    def insert(self, i, s):
        TextWithMarks.insert(self, i, s)
        self.UpdateTags(i, len(s))
```

9 Text

Purpose

This class provides a wrapper around the class TextWithMarksTags, which allows positions in the text to be given as (line, column) pairs. It will only forward all the method calls to the corresponding method calls on the component class instance of TextWithMarksTags, after translating the indexes to positions (and the other way, when needed).

Structure




```
class Text:
    def __init__(self):
        self.text := TextWithMarksTags(ElementWithTags)
    def __invariant__(self):
        self.text.__invariant__()
    def index(self, pos):
        l, c, p := 1, 0, 0
        while p < pos:
            if self.text[p] = '\n':
                l, c := l+1, 0
            else:
                c := c + 1
            p := p + 1
        return l, c
    def position(self, index):
        if index = None:
            return
        line, col := index
        l, pos := 1, 0
        while l < line and pos < self.text.END():
            if self.text[pos] = '\n':
                l := l + 1
            pos := pos + 1
        return pos + col
    def END(self):
        line, col := self.index(self.text.END())
        return line+1, 0
    def positionOK(self, index):
        return self.text.positionOK(self.position(index))
    def insert(self, index, s):
        self.text.insert(self.position(index), s)
    def delete(self, startindex, endindex:=None):
        self.text.delete(self.position(startindex), self.position(endindex))
    def get(self, startindex, endindex:=None):
        return self.text.get(self.position(startindex), self.position(endindex))
```

Comment

Only part of the methods are described above. All methods are implemented in the same way, by forwarding the method call to the attribute text.

The position method is incomplete: need to decide what position to return when index is illegal (line outside permitted range, or column outside permitted range). Tkinter returns closest reasonable position.

We consider an index here simply as a (line, column) pair.

Analysis

We need to convince ourselves that the co-ordinate transformation between indexes and positions is correct. We do this in two stages. First, we show that the two methods implement a more abstract specification, and then we show that the abstract specifications have desired properties.

The function split(str, '\n') will split the sequence str into a sequence of subsequences, such that when the subsequences are joined together with '\n' between, we get the original sequence. (This is really defined for strings, need a little bit work to define it for sequences of elements).

The predicate indexPosition(str, pos, line, col) expresses the correspondence between position and (line, column) index in the strings str. We define it by

```
def indexPosition(str, pos, line, col):
    s = split(str, '\n')
    return (0≤pos≤len(str)) and (1≤line≤len(s)) and
           (0≤col≤len(s[line-1])) and (∑_{i=0}^{line-2}(len(s[i]+1)) + col = pos)
```

Observation. If str is fixed then the relation *indpos* : Nat ↔ Nat × Nat given by *indpos*(pos, line, col) ≡ indexPosition(str, pos, line, col) is an injective partial function.

We can now define the operation position with the help of a nondeterministic assignment statement:

```
def position(self, (line, col)):
    pos := pos. indexPosition(self.text, pos, line, col)
    return pos
```

Similarly, we define the index method by

```
def index(self, pos):
    line, col := (line, col). indexPosition(self.text, pos, line, col)
    return line, col
```

The following features can now be analyzed:

1. The method position is a refinement of the specification above.
2. The method index is a refinement of the specification above.
3. First calculating the position and then the index from the resulting position gives the original index.
4. First calculating the index and then the position from the resulting index gives the original index.

The last two properties establish that position and index are both bijections, i.e., that there is a one-to-one correspondence between positions and legal indexes.

Index

We sould also define operations on text indexes, to model the way in which one can manipulate indexes in Tkinter (there considered to be string of a particular form). The basic operations for indexes are

- get the start of line index for a given index
- get the end of line index for a given index
- move the index one position right(may change line)
- move the index one position left
- move the index one line before (keeping column, if possible)
- move the index one line after (keeping column, if possible)

Note that all these operations require that we know the text string, hence they are methods of the class Text, rather than operations of an independet class Index. As an example, we show how to define the operation for moving the index one position right:

```
def indextRight(self, index):
    pos := self.position(index)
    pos := pos+1
    return self.index(pos)
```

This definition does not take into account the possibility that we would move the index past the last position. Need to add guards against this kind of conditions.

10 An implementation

Introducing one feature at a time leads to a big collection of classes. This is useful from designing point of view because dealing with only one feature is much easier. But it is not good if we want to implement a real object.

In this section we show how one can make a single object that contains all features of the classes described above. The method is to replace the procedure call with the body of the procedure in the bottom object from the Figure 1. This operation is rather mechanical and can be done by a program.

We only show for the insert and delete methods of the class TextWithMarksTags. All other functions from TextWithMarksTags can be transformed in the same way. Using this method all objects from Figure 1 can be transformed into a single object that has all features.

TextWithMarksTags.insert:

```
    TextWithMarksTags.insert(self, i, s)
=
    TextWithMarks.insert(self, i, s)
    self.UpdateTags(i, len(s))
=
    TextWithElements.insert(self, i, s)
    self.UpdateMarks(i, len(s))
    self.UpdateTags(i, len(s))
```

=

```
assert self.positionOK(i) and IsString(s)
text := self.makeText(s)
self.str := self.str[:i] + text + self.str[i:]
self.UpdateMarks(i, len(s))
self.UpdateTags(i, len(s))
```

TextWithMarksTags.delete:

```
TextWithMarksTags.delete(self, i, j)
```

=

```
TextWithMarks.delete(self, i, j)
```

=

```
if j=None:
    j := i + 1
TextWithElements.delete(self, i, j)
for m in self.mark.keys():
    if self.mark[m]['pos'] in range(i, j):
        self.mark[m]['pos'] := i
    elif self.mark[m]['pos'] ≥ j:
        self.mark[m]['pos'] := self.mark[m]['pos'] - (j-i)
```

=

```
if j=None:
    j := i + 1
if j=None:
    j := i + 1
assert self.positionOK(i) and self.positionOK(j) and (i ≤ j)
self.str := self.str[:i] + self.str[j:]
for m in self.mark.keys():
    if self.mark[m]['pos'] in range(i, j):
        self.mark[m]['pos'] := i
    elif self.mark[m]['pos'] ≥ j:
        self.mark[m]['pos'] := self.mark[m]['pos'] - (j-i)
```

=

```
if j=None:
    j := i + 1
assert self.positionOK(i) and self.positionOK(j) and (i ≤ j)
self.str := self.str[:i] + self.str[j:]
for m in self.mark.keys():
    if self.mark[m]['pos'] in range(i, j):
        self.mark[m]['pos'] := i
    elif self.mark[m]['pos'] ≥ j:
        self.mark[m]['pos'] := self.mark[m]['pos'] - (j-i)
```

11 Testing

To test the specification we should show two things:

1. The addition of a new feature to a given class of the specification is made preserving its functionality;

2. Any class from the specification that implements some feature of the text-widget should behave as the real feature of it.

We have tested the internal consistency of a class derivation by writing a function that takes as parameter an instance of the base class or of the derived class and test the features implemented by the base class.

```
def Test_TextWithMarks(x):
    Test_TextWithElements(x)
    x.delete(0, x.END())
    x.insert(0,'12345')
    x.mark_set('m', x.END())
    assert(x.index('m') == x.END())
    assert(x.mark_gravity('m') == 'right')
    x.delete(1)
    assert(x.index('m') == x.END())
    x.insert(x.END(),'oooo')
    assert(x.index('m') == x.END())
    i = x.END()
    x.mark_gravity('m', LEFT)
    x.insert(x.END(),'AAAAA')
    assert(x.index('m') == i)
    x.mark_set('m1', 0)
    x.insert(0,'xxxxxxxx')
    assert(x.index('m1') == 8)
    x.delete(1, 3)
    assert(x.index('m1') == 6)
    x.delete(4,8)
    assert(x.index('m1') == 4)
    x.delete(6,9)
    assert(x.index('m1') == 4)
    assert(x.mark_names() == ['current', 'insert', 'm', 'm1'])
    x.mark_unset('m1')
    assert(x.mark_names() == ['current', 'insert', 'm'])
    x.mark_unset('m')
    assert(x.mark_names() == ['current', 'insert'])

Test_TextWithMarks(TextWithMarks(Element))
Test_TextWithMarks(TextWithMarksTags(ElementWithTags))
```

We have not tested that the specification behaves in the same way as the real Text-widget. We could use the same approach as before. To be able to do so we only need to implement indexes used by Text-widget, i.e. “line.column”. Then we could write a function that can be applied to an instance of the text-widget or to an instance of the specification and returns a string with the output of the test. Then the two strings must be equals.

12 Conclusions

We believe that such kind of specification/documentation is very useful when writing programs. Usually when a programmer does not find the answer for his question by reading the documentation he/she goes to the source code and tries to see how it really works. Our approach is to give the source code in a more systematic way. This specification can be executed, tested. It is not intended to replace the real code that could be more efficient.

This approach can be used too for specifying a program. After the program implementation the specification can be used as its documentation.

References

[1] Ralph-Johan Back. Software Construction by Stepwise Feature Introduction. To appear in *Proceedings of the ZB2001 - Second International Z and B Conference*, Springer Verlag LNCS Series, 2002.

[2] John E. Grayson. *Python and Tkinter Programming.* Manning, 2000.

[3] Fredrik Lund. An introduction to tkinter. Technical report, 1999.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1338-8
ISSN 1239-1891