Traducer textos con Gtk TextView

Le texto

Nostre texto es le initio de *Vita in le foreste* (1854) per Henry David Thoreau.

```
When I wrote the following pages, or rather the bulk of them, I
lived alone, in the woods, a mile from any neighbor, in a house
which I had built myself, on the shore of Walden Pond, in
Concord, Massachusetts, and earned my living by the labor of my
hands only. I lived there two years and two months. At present I
am a sojourner in civilized life again.
```

Nos proba traducer le texto. Illo comencia: "*Quando io scribeva le paginas sequente, o …*"

Expressiones regular

Nos usa un *expression regular*

```
re = "[[:alpha:]]+"
```

Ma prime nos vide como functiona un expression regular. Per exemplo, le expression regular `"e"` accepta le littera `e`. Ecce altere exemplos.

`"e"` littera `e`

`"[eaio]"`
: un de litteras `e`, `a`, `i`, `o`

`"e*"`
: littera `e` zero o plure vices

`"e+"`
: littera `e` un o plure vices

`"[eaio]+"`
: iste litteras un o plure vices, per exemplo: `"aeo"`, `"eei"`, `"i"`, `"iiiia"`, `"oaoao"`, …

`"[[:alpha:]]"`
: littera ex alphabeto

`"[[:alpha:]]+"`
: litteras ex alphabeto un o plure vices

Characteres special es `".[{()\*+?|^$"`. Alteremente un character significa le character ipse.

Parolas

Nos trova que le expression regular `"[[:alpha:]]+"` significa un *parola* (parola false o parola correcte) (Figura 1).

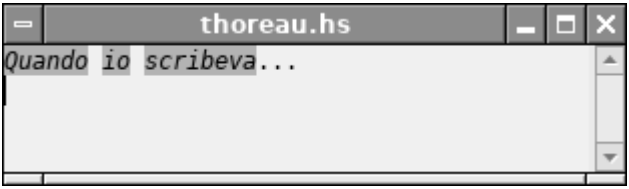

Figura 1. Parolas in texto.

Gtk `TextView` componente

In le programma, nos usa le componente `TextView`. Un `TextView` monstra un texto ex `TextBuffer`. Un `TextBuffer` ha un tabula `TextTagTable`, qui es un collection de objectos `TextTag`.

Un `TextTag` ha un *location de initio* e un *location de fin*. Un `TextTag` anque pote haber un *description* con *attributos* e un *nomine*.

Le attributos de un `TextTag` pote esser per exemplo le typo, le color e le grandor de litteras.

In iste programma nos usa objectos de typo `TextTag` pro cambiar le color de fundo (a gris) e le typo de litteras (a oblique). Le limites nos trova usante le expression regular pro parolas.

```
import Graphics.UI.Gtk
import Text.Regex.Posix
import Text.Regex
```

```
import Text.Regex.Base
import Data.Array
import System.Glib.UTFString

re = "[[:alpha:]]+"
reX = makeRegex re :: Regex

matchesA2 value = map elems $ matchAll reX (value :: String)
matchesA3 value = map assocs $ matchAll reX (value :: String)
matchesA4 value = map (!0) $ matchAll reX (value :: String)

subsK value = subRegex reX value "*\\0*"

grays = map gray [0.7,0.6..0.0]

createTag table n = do
  tag <- textTagNew (Just (stringToGlib ("gray" ++ show n)))
  let fg = if n > length grays `div` 2
             then head grays else last grays
  set tag [
    textTagStyle := StyleItalic,
    textTagForegroundGdk := fg,
    textTagBackgroundGdk := grays !! n]
  textTagTableAdd table tag

createTags table = do
 mapM (createTag table) [0 .. length grays - 1]

textBufferGetValue buffer = do
  start <- textBufferGetStartIter buffer
  end <- textBufferGetEndIter buffer
  value <- textBufferGetText buffer start end True
  return value

paintTag buffer ((a,b),n) = do
  iter0 <- textBufferGetIterAtOffset buffer a
  iter1 <- textBufferGetIterAtOffset buffer (a+b)
```

```
  textBufferApplyTagByName buffer
    (stringToGlib ("gray" ++ show n)) iter0 iter1

paintTags buffer xs = do
  mapM_ (paintTag buffer) (zip xs [0 .. length grays - 1])

main = do
  initGUI
  content <- readFile "thoreau-initio.txt"
  window <- windowNew
  sw <- scrolledWindowNew Nothing Nothing
  set sw [
    scrolledWindowVscrollbarPolicy := PolicyAlways,
    scrolledWindowHscrollbarPolicy := PolicyAutomatic ]
  view <- textViewNew
  buffer <- textViewGetBuffer view
  table <- textBufferGetTagTable buffer
  font <- fontDescriptionFromString "Monospace 9"
  widgetModifyFont view (Just font)
  widgetModifyBase view StateNormal (gray 0.94)
  textBufferSetText buffer content
  containerAdd (toContainer sw) view
  set window [
    windowDefaultWidth := 310,
    windowDefaultHeight := 160,
    containerChild := sw]
  on window objectDestroy mainQuit
  widgetShowAll window
  value <- textBufferGetValue buffer
  createTags table
  let xs = matchesA2 value
  mapM_ (paintTags buffer) xs
  putStrLn "TEXT = "
  putStrLn value
  putStrLn "RE = "
  putStrLn re
  putStrLn "A2 = "
```

```
  print (matchesA2 value)
  putStrLn "A3 = "
  print (matchesA3 value)
  putStrLn "A4 = "
  print (matchesA4 value)
  putStrLn "K = "
  print (subsK value)
  mainGUI

gray n = Color gt gt gt
  where
    gt = round (n * 65535)
```

Le resultatos de programma es

```
TEXT = Quando io scribeva...

RE = [[:alpha:]]+

A2 = [[(0,6)],[(7,2)],[(10,8)]]

A3 = [[(0,(0,6))],[(0,(7,2))],[(0,(10,8))]]

A4 = [(0,6),(7,2),(10,8)]

K = "*Quando* *io* *scribeva*...\n"
```

Libreria `Graphics.UI.Gtk.Multiline.TextView`

Un `TextView` monstra un texto ex `TextBuffer`.

Typos:
`TextView MovementStep DirectionType TextWindowType TextChildAnchor WrapMode DeleteType Justification`

Prefixo: `textView-`
Constructores: `~New ~NewWithBuffer`

Methodos:
```
~SetBuffer ~MoveMarkOnscreen ~AddChildAtAnchor ~BackwardDisplayLineStart
~GetBuffer  ~GetIterLocation ~AddChildInWindow textChildAnchorGetWidgets
~GetWindow   ~SetRightMargin ~SetCursorVisible textChildAnchorGetDeleted
~MoveChild   ~GetRightMargin ~GetCursorVisible  ~ImContextFilterKeypress
~SetIndent   ~GetVisibleRect ~SetJustification    ~ForwardDisplayLineEnd
```

```
~GetIndent   ~GetHadjustment ~GetJustification     ~BufferToWindowCoords
~GetLineAtY  ~GetVadjustment ~GetIterAtLocation    ~WindowToBufferCoords
~SetWrapMode ~ResetImContext ~StartsDisplayLine    ~GetDefaultAttributes
~GetWrapMode  ~GetLineYrange textChildAnchorNew     ~BackwardDisplayLine
~SetEditable  ~GetWindowType ~GetIterAtPosition     ~SetPixelsAboveLines
~GetEditable  ~SetLeftMargin ~ScrollMarkOnscreen    ~GetPixelsAboveLines
~ScrollToMark ~GetLeftMargin ~ForwardDisplayLine    ~SetPixelsBelowLines
~ScrollToIter ~SetAcceptsTab ~PlaceCursorOnscreen   ~GetPixelsBelowLines
~MoveVisually ~GetAcceptsTab ~SetBorderWindowSize   ~SetPixelsInsideWrap
~SetOverwrite  ~GetOverwrite ~GetBorderWindowSize   ~GetPixelsInsideWrap
```

Attributos:
```
~Indent  ~Overwrite ~LeftMargin ~PixelsAboveLines ~PixelsBelowLines
~Buffer   ~ImModule ~AcceptsTab    ~Justification ~PixelsInsideWrap
~Editable ~WrapMode ~RightMargin   ~CursorVisible
```

Signales:
```
backspace copyClipboard insertAtCursor set~ScrollAdjustments
moveFocus populatePopup pasteClipboard   toggleCursorVisible
selectAll  cutClipboard toggleOverwrite     deleteFromCursor
setAnchor  moveViewport ~PreeditChanged     pageHorizontally
moveCursor
```

Libreria `Graphics.UI.Gtk.Multiline.TextBuffer`

Un `TextBuffer` contine le texto con su attributos.

Typos: `TextBuffer`
Prefixo: `textBuffer-`
Constructores: `~New`
Methodos:
```
~Insert   ~GetLineCount ~RemoveAllTags ~InsertInteractiveAtCursor
~Delete   ~GetCharCount ~GetIterAtLine  ~RemoveSelectionClipboard
~SetText  ~InsertPixbuf ~GetIterAtMark    ~InsertRangeInteractive
~GetText  ~GetStartIter ~EndUserAction    ~PasteClipboardAtCursor
~AddMark  ~HasSelection ~CopyClipboard     ~AddSelectionClipboard
~GetMark  ~CutClipboard ~InsertAtCursor     ~GetIterAtChildAnchor
~GetSlice  ~GetTagTable ~MoveMarkByName      ~GetIterAtLineOffset
~MoveMark  ~InsertRange ~ApplyTagByName       ~GetSelectionBounds
~ApplyTag  ~PlaceCursor ~PasteClipboard        ~InsertInteractive
~GetInsert ~GetModified ~RemoveTagByName       ~DeleteInteractive
~RemoveTag ~SetModified ~GetIterAtOffset       ~GetSelectionBound
~GetBounds ~SelectRange ~DeleteSelection       ~InsertChildAnchor
~Backspace  ~DeleteMark ~BeginUserAction       ~CreateChildAnchor
~CreateMark ~GetEndIter ~DeleteMarkByName
```

Attributos: `~TagTable ~Text ~Modified`

Signales:
```
markSet  deleteRange insertPixbuf  beginUserAction bufferInsertText
```

```
applyTag markDeleted bufferChanged modifiedChanged insertChildAnchor
pasteDone  removeTag endUserAction
```

Libreria `Graphics.UI.Gtk.Multiline.TextTagTable`

Un `TextTabTable` es un collection de objectos `TextTag`.

Typos: `TextTagTable`
Prefixo: `textTagTable-`
Constructores: `~New`
Methodos: `~Add ~Remove ~Lookup ~Foreach ~GetSize`

Libreria `Graphics.UI.Gtk.Multiline.TextTag`

Un `TextTag` ha un location de initio e un location de fin (ambes es componentes `TextIter`). Un `TextTag` anque pote haber un description con attributos e un nomine.

Le attributos de un `TextTag` pote esser per exemplo le typo, le color e le grandor de litteras.

Nos pote *applicar* un `TextTag` pro facer le attributos visibile o effectuar illos.

Nos pote usar functiones `textBufferGetTagTable`, `textTagNew` e `textTagTableAdd` pro crear componentes `TextTag`.

Typos: `TextTag TagName`
Prefixo: `textTag-`
Constructores: `~New`
Methodos: `~SetPriority ~GetPriority makeNewTextAttributes`
`textAttributesNew textAttributesCopy textAttributesCopyValues`

Attributos:

```
~Name   ~Background ~LeftMargin ~BackgroundFullHeightSet
~Font   ~Foreground ~EditableSet ~ParagraphBackgroundSet
~Size   ~VariantSet ~LanguageSet ~ParagraphBackgroundGdk
~Rise   ~StretchSet ~RightMargin   ~BackgroundFullHeight
~Style  ~SizePoints ~WrapModeSet   ~BackgroundStippleSet
~Scale   ~Direction ~UnderlineSet  ~ForegroundStippleSet
~Family  ~FamilySet ~InvisibleSet   ~PixelsAboveLinesSet
~Weight  ~WeightSet ~BackgroundSet  ~PixelsBelowLinesSet
~Indent  ~IndentSet ~BackgroundGdk  ~PixelsInsideWrapSet
~TabsSet ~Underline ~ForegroundSet  ~ParagraphBackground
~Variant ~Invisible ~ForegroundGdk    ~BackgroundStipple
~Stretch  ~StyleSet ~Justification    ~ForegroundStipple
```

```
~SizeSet  ~ScaleSet ~LeftMarginSet     ~PixelsAboveLines
~RiseSet  ~Language ~Strikethrough     ~PixelsBelowLines
~Editable ~WrapMode ~RightMarginSet    ~PixelsInsideWrap
~FontDesc ~Priority ~JustificationSet  ~StrikethroughSet
```

Signales: `onTextTagEvent`

Libreria `Graphics.UI.Gtk.Multiline.TextIter`

Un `TextIter` es un location in `TextBuffer`.

Typos: `TextIter TextSearchFlags`
Prefixo: `textIter-`
Methodos:

```
~Copy   ~GetLineOffset ~SetLineOffset ~BackwardVisibleCursorPositions
~IsEnd  ~GetAttributes ~ForwardSearch  ~BackwardVisibleCursorPosition
~Equal  ~BackwardChars ~GetVisibleText ~ForwardVisibleCursorPositions
~Order  ~BackwardLines ~GetChildAnchor  ~ForwardVisibleCursorPosition
~HasTag  ~EndsSentence ~GetToggledTags     ~BackwardVisibleWordStarts
~GetLine ~BackwardChar ~StartsSentence      ~BackwardVisibleWordStart
~GetChar ~ForwardChars ~InsideSentence       ~BackwardCursorPositions
~GetText ~BackwardLine ~GetCharsInLine        ~BackwardCursorPosition
~EndsTag ~ForwardLines ~ForwardWordEnd        ~ForwardCursorPositions
~GetTags ~ForwardToEnd ~BackwardSearch        ~BackwardSentenceStarts
~IsStart  ~GetLanguage ~GetVisibleSlice       ~ForwardVisibleWordEnds
~SetLine  ~ForwardChar ~ForwardWordEnds        ~ForwardCursorPosition
~Compare  ~ForwardLine ~ForwardFindChar        ~BackwardSentenceStart
~InRange   ~TogglesTag ~IsCursorPosition       ~ForwardVisibleWordEnd
~GetSlice  ~StartsWord ~ForwardToLineEnd        ~GetVisibleLineOffset
~GetMarks  ~InsideWord ~BackwardFindChar        ~SetVisibleLineOffset
~Editable  ~StartsLine ~BackwardWordStart       ~BackwardVisibleLines
~EndsWord   ~GetPixbuf ~BackwardWordStarts       ~ForwardSentenceEnds
~EndsLine   ~BeginsTag ~ForwardSentenceEnd       ~BackwardToTagToggle
~GetBuffer  ~CanInsert ~ForwardToTagToggle       ~BackwardVisibleLine
~GetOffset  ~SetOffset ~ForwardVisibleLine       ~ForwardVisibleLines
```

Attributos: `~VisibleLineOffset ~Offset ~LineOffset ~Line`

Libreria `Graphics.UI.Gtk.Multiline.TextMark`

Un `TextMark` es un location in `TextBuffer`, que conserva su location correcte quando on manipula le texto in `TextBuffer`.

Isto es al contrario de un `TextIter`, que non remane valide post manipulation de texto.

On pote converter un `TextMark` a `TextIter` per le function `textBufferGetIterAtMark`.

Un `TextMark` pote haber un nomine.

On pote crear un `TextMark` per le function `textBufferCreateMark`.

Typos: `TextMark MarkName`
Prefixo: `textMark-`
Constructores: `~New`
Methodos: `~SetVisible ~GetVisible ~GetDeleted ~GetName ~GetBuffer ~GetLeftGravity`
Attributos: `~Name ~Visible ~LeftGravity`

Cognoscer le arbores

Nos proba un altere texto, un tabula con arbores de folios (Figura 2).

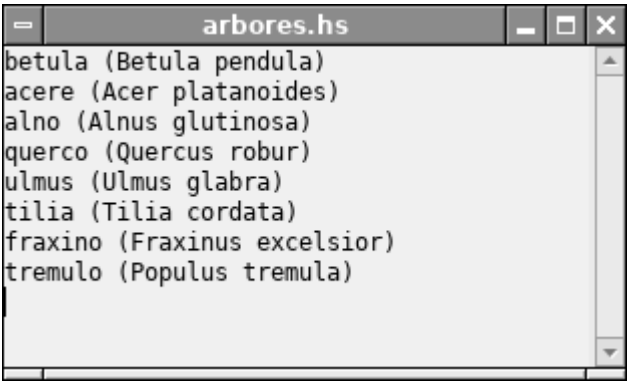

Figura 2. Texto con arbores.

```
betula (Betula pendula)
acere (Acer platanoides)
alno (Alnus glutinosa)
querco (Quercus robur)
ulmus (Ulmus glabra)
tilia (Tilia cordata)
fraxino (Fraxinus excelsior)
tremulo (Populus tremula)
```

Ecce nos lege le texto:

```
content <- readFile "arbores.txt"
```

Nos defini le function `slice`.

```
slice xs (start,n) = take n (drop start xs)
```

Ecce un resultato de nostre function:

```
> slice "eaionlrstucdpm" (2,3)
"ion"
```

Le function `toLower` ex librario `Data.Char` converte un littera a minuscule.

```
> import Data.Char
> map toLower "VitaInLeForeste"
"vitainleforeste"
```

Nos adjusta le programma.

```
value <- textViewGetValue view
let
  xs = matchesA2 value
  ts = matchesA4 value
  ws1 = map (slice value) ts
  ws = map (map toLower) ws1
putStr "ts = "
print ts
putStr "ws = "
print ws
```

Le resultatos de programma es

```
ts = [(0,6),(8,6),(15,7),(25,5),(32,4),(37,11),(50,4),(56,5),
(62,9),(73,6),(81,7),(89,5),(96,5),(103,5),(109,6),(117,5),
(124,5),(130,7),(139,7),(148,8),(157,9),(168,7),(177,7),(185,7)]

ws = ["betula","betula","pendula","acere","acer","platanoides",
"alno","alnus","glutinosa","querco","quercus","robur","ulmus",
"ulmus","glabra","tilia","tilia","cordata","fraxino","fraxinus",
"excelsior","tremulo","populus","tremula"]
```

Nos usa le librario `Data.Map`

```
import qualified Data.Map as Map
```

Nos lege alicun parolas ex dictionario `"51500-ii.txt"` e face de illes un *arbore de cercar* (`Map`).

```
dictText <- readFile "51500-ii.txt"
let
  dictWs = lines dictText
  dictTree = Map.fromList [(map toLower w,"") | w <- dictWs]
```

Le file `"51500-ii.txt"` ha le formato (ubi nos trova, que nostre *ulmus* deberea esser *ulmo*)

```
...
ulmeto
ulmifolie
ulmo
ulna
ulnar
Ulster
ulterior
...
```

Nos ha le parolas de `TextBuffer` in lista `ws` e le parolas de dictionario in arbore `dictTree`. Nos nunc pote demandar si le cata parola de `TextBuffer` existe in arbore `dictTree`, id es, si illes es *membros* (`Map.member`) de arbore.

```
let
  xs = matchesA2 value
  ts = matchesA4 value
  ws = map (slice value) ts
  ks = map (`Map.member` dictTree) ws
  ys = zip3 ts ws ks
mapM_ (paintTag3 buffer) ys
putStr "ks = "
print ks
```

Le lista `ks` es

```
ks = [True,True,True,True,False,False,True,False,False,True,
False,False,False,False,False,True,True,True,True,False,True,
True,False,False]
```

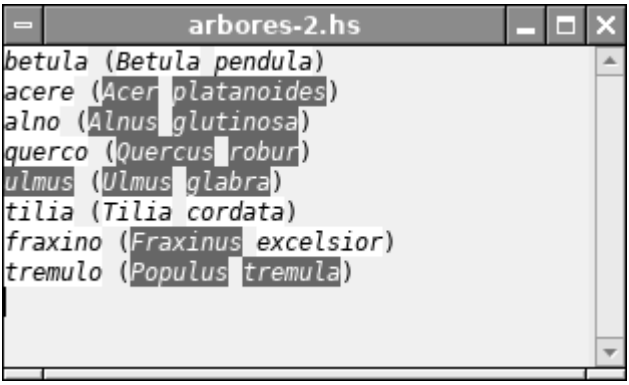

Figura 3. Texto con nomines de arbores valide e non-valide.

Hic valor `True` significa, que parola es membro de arbore `dictTree`, e valor `False` que illo non es. Nos pinge cata parola correspondente in `TextView`.

```
paintTag3 buffer ((a,b),name,member)  = do
  let n = if member then 0 else 6
  paintTag buffer ((a,b),n)
```

La algorithmo remane le mesmo.

```
grays = map gray [1.0,0.9..0.0]

paintTag buffer ((a,b),n) = do
  iter0 <- textBufferGetIterAtOffset buffer a
  iter1 <- textBufferGetIterAtOffset buffer (a+b)
  textBufferApplyTagByName buffer
    (stringToGlib ("gray" ++ show n)) iter0 iter1
```

Nos scribe un tabula de resultatos anque.

```
let table = tabula2 ws ks
mapM_ putStrLn table

prolongar2 n x y = x ++ replicate r ' ' ++ y
```

```
  where
    l = length x + length y
    r = n - l

tabula1 xs = res1
  where
    res1 = [prolongar2 n a b | (a,b) <- xs]
    n = maximum [length a + length b + 1 | (a,b) <- xs]

tabula2 ws ks = map (intercalate " ") res2
  where
    res2 = transpose cs
    cs = map tabula1 rows
    rows = chunksOf c xs
    c = l `div` cols + if (l `mod` cols > 0) then 1 else 0
    l = length xs
    xs = zip ws ms
    ms = map show ks
    cols = 4
```

Le tabula es

```
betula       True alno       True ulmus  False fraxino   True
betula       True alnus     False ulmus  False fraxinus False
pendula      True glutinosa False glabra False excelsior True
acere        True querco     True tilia   True tremulo   True
acer        False quercus   False tilia   True populus  False
platanoides False robur     False cordata True tremula  False
```

Nos scribe le parolas que nos trova e le parolas que nos non trova.

```
let
  fnd = intercalate " " [a |(a,b) <- zip ws ks, b]
  notfd = intercalate " " [a |(a,b) <- zip ws ks, not b]
putStrLn ("fnd = " ++ fnd)
putStrLn ("notfd = " ++ notfd)
```

Resultato es

```
fnd = betula betula pendula acere alno querco tilia tilia cordata
fraxino excelsior tremulo
```

```
notfd = acer platanoides alnus glutinosa quercus robur ulmus ulmus
glabra fraxinus populus tremula
```

Alterationes de text

Quando le scriptor altera le texto, componente `TextBuffer` emitte un signal `bufferChanged`. Nos responde al signal per definir un function `bfChanged`.

```
on buffer bufferChanged $ bfChanged buffer


bfChanged buffer = do
  liftIO $ putStr "bfChanged. Offset = "
  insMark <- textBufferGetInsert buffer
  insIter <- textBufferGetIterAtMark buffer insMark
  offset <- textIterGetOffset insIter
  liftIO $ print offset
```

Le programma ecce annuncia le alterationes.

```
bfChanged. Offset = 100
bfChanged. Offset = 99
bfChanged. Offset = 100
```

Nos non vole responder a cata alteration, ma solmente quando le scriptor pausa un momento. Pro isto nos besonia un componente `Timeout`. Nos trova un `Timeout` in libreria `Graphics.UI.Gtk.General.General`.

Libreria `Graphics.UI.Gtk.General.General`

Libreria `Graphics.UI.Gtk.General.General` ha le functiones sequente:

```
initGUI postGUISync priorityLow grabGetCurrent timeoutAddFull
mainGUI mainDoEvent inputRemove  eventsPending mainIterationDo
grabAdd  grabRemove postGUIAsync mainIteration priorityDefault
idleAdd  timeoutAdd threadsEnter timeoutRemove priorityHighIdle
mainQuit idleRemove threadsLeave  priorityHigh priorityDefaultIdle
inputAdd  mainLevel
```

Functiones `timeoutAdd` e `timeoutRemove` es utile a crear e remover componentes `Timeout`.

Crear un `Timeout`

Nos crea un `Timeout` per adder function `timeoutAdd`. Nos defini le function `timeoutFunc`, qui responde a signales de nostre componente `Timeout`.

```
timeoutAdd timeoutFunc 1000

timeoutFunc = do
  liftIO $ putStr "tio "
  return True
```

Le programma scribe cata secundo le parola `"tio"` (pro `timeout`).

```
tio tio tio tio tio bfChanged. Offset = 113
tio tio tio tio tio tio tio tio tio tio
```

Calcular secundas

Pro delivrar un stato, nos usa un `MVar`. Nos delivra le secundos calculate e le stato si nos debe pinger le parolas.

```
var <- newMVar (9,Changed)

paintBuffer var buffer dictTree = do
  (old,o) <- readMVar var
  let
    new = 0
    oo = Painted
  modifyMVar_ var (\_ -> return (new,oo))
  paintBuffer1 buffer dictTree
  liftIO $ print ("painted",(new,oo))
  return ()

timeoutFunc var buffer dictTree = do
```

```
  (old,o) <- readMVar var
  let new = old + 1
  modifyMVar_ var (\_ -> return (new,o))
  if (new >= 10 && o == Changed)
    then paintBuffer var buffer dictTree
    else return ()
  liftIO $ print ("tio",(new,o))
  return True
```

Post alteration, nostre programma attende 10 secundas e pinge le parolas (Figura 4). Ancora nostre programma non cognosce plurales o formas de verbos.

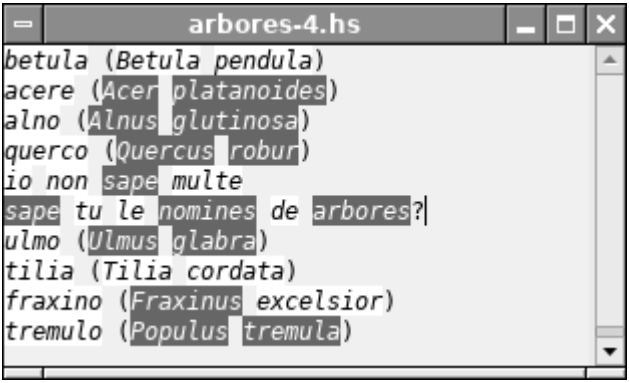

Figura 4. Texto post alteration.

Le programma annuncia su calculationes.

```
("tio",(1,Changed))
bfChanged. Offset = 144
("tio",(1,Changed))
("tio",(2,Changed))
("tio",(3,Changed))
("tio",(4,Changed))
("tio",(5,Changed))
("tio",(6,Changed))
("tio",(7,Changed))
("tio",(8,Changed))
```

```
("tio",(9,Changed))
("painted",(0,Painted))
("tio",(10,Changed))
("tio",(1,Painted))
```

Libreria `Data.Map`

Libreria `Data.Map` ha functiones de un arbore de cercar.

```
(!)  lookupLT lookupGE  unionWithKey showTreeWith  traverseMaybeWithKey
map  lookupGT mapAccum  intersection insertWithKey fromDistinctDescList
(!?) lookupLE fromList  mergeWithKey adjustWithKey  insertLookupWithKey
( )  mapKeys mapMaybe  foldrWithKey updateWithKey  updateLookupWithKey
null  keysSet updateAt  foldlWithKey foldrWithKey'  intersectionWithKey
size  fromSet deleteAt  fromListWith foldlWithKey'  fromDistinctAscList
keys  splitAt showTree  fromDescList filterWithKey  fromDescListWithKey
take  findMin notMember restrictKeys deleteFindMin   fromAscListWithKey
drop  findMax singleton spanAntitone deleteFindMax   isProperSubmapOfBy
empty minView unionWith isSubmapOfBy differenceWith   differenceWithKey
alter maxView toAscList  mapKeysWith foldMapWithKey   takeWhileAntitone
union  adjust partition  fromAscList minViewWithKey   dropWhileAntitone
foldr  update mapEither  withoutKeys maxViewWithKey    mapAccumRWithKey
foldl  alterF splitRoot  splitLookup findWithDefault   mapKeysMonotonic
elems  unions findIndex  lookupIndex traverseWithKey   fromDescListWith
split  foldr' lookupMin   insertWith mapAccumWithKey   partitionWithKey
valid  foldl' lookupMax   unionsWith fromListWithKey   mapEitherWithKey
member assocs deleteMin   difference fromAscListWith   isProperSubmapOf
lookup toList deleteMax   mapWithKey mapMaybeWithKey   updateMinWithKey
insert filter updateMin   toDescList intersectionWith  updateMaxWithKey
delete elemAt updateMax   isSubmapOf
```

Regulas e parolas

Nos ha le file `51500-tab.txt`, que ha le formato sequente. Le partes ha tabulatores `'\t'` inter se.

```
...
fossile adj
fossile sb
fossilifere sb
fossilisar  vb
fossilisation   sb
fossor  sb
```

```
fossori adj
foulard sb
fovea   sb
...
```

Nos scribe un file `regulas-5.txt`, ubi nos monstra como formar le modo basic ex substantivos plural e ex verbos in tempores presente, participio presente, participio passate e tempore passate. Anque hic le partes ha tabulatores `'\t'` inter se.

```
(.*[eaiouy])s   \1  sb (plural)
(.*[^eaiouyc])es    \1  sb (plural)
(.*)ches    \1c sb (plural)
(.*)a   \1ar    vb (tempore presente)
(.*)e   \1er    vb (tempore presente)
(.*)i   \1ir    vb (tempore presente)
(.*)ante    \1ar    vb (participio presente)
(.*)ente    \1er    vb (participio presente)
(.*)iente   \1ir    vb (participio presente)
(.*)ate \1ar    vb (participio passate)
(.*)ite \1er    vb (participio passate)
(.*)ite \1ir    vb (participio passate)
(.*)va  \1r vb (tempore passate)
```

Pro isto nos besonia alicun nove regulas de expressiones regular.

`"^"` initio

`"$"` fin

`"[^eai]"`
: nemo de litteras `e`, `a`, `i`

`"()"`
: un gruppo

`"\0"`
: gruppo numero zero, id es, plen texto trovate.

`"\1"`
: prime gruppo, id es, texto in prime parenthesis

Quando nos pone un texto in parentheses, nos forma un gruppo. Cata gruppo ha un numero, que nos pote usar pro cambiar illo.

Nos lege un texto exemplar, le dictionario e le regulas

```
content <- readFile "thoreau-initio-2.txt"
dictText <- readFile "51500-tab.txt"
rulesText <- readFile "regulas-5.txt"
```

Nos forma ex dictionario un arbore de cercar.

```
createWs2 dictText = ds3
  where
    ls = lines dictText
    ds1 = [breakTab w | w <- ls]
    ds2 = filter (\(a,b) -> isWord a) ds1
    ds3 = Map.fromList ds2

breakTab w = (map toLower a, words (drop 1 b))
  where
    (a,b) = break (=='\t') w

isWord w = w == takeWhile isAlpha w
```

Nos usa le regulas pro trovar le formo basic de parola.

```
ruleEtExpl [r,repl,expl] w =
  if res1 /= w then (res1, expl2) else ("","")
  where
    rplus = "^" ++ r ++ "$"
    reX = makeRegex rplus :: Regex
    res1 = subRegex reX w repl
    expl2 = "→ " ++ w ++ " "++ expl
ruleEtExpl _ _ = ("","")

ruleEtExpls w rs = filter (/= ("","")) [ruleEtExpl r w | r <- rs]
```

Le function `matchingWds` cerca le parola in arbore de cercar `tree` usante regulas `rules`, que nos dava supre.

```
matchingWds tree rules wd =
  [ (w,x) | (w,x) <- wrs, w `Map.member` tree  ]
  where
```

```
    wd1 = map toLower wd
    vrs = ruleEtExpls wd1 rules
    wrs = [(wd1,w2)] ++ vrs
    wr =  Map.lookup wd1 tree
    w1 = fromMaybe [] wr
    w2 = intercalate " " w1
```

Crear componentes `TextView`

Nos vole crear alicun componentes de typo `TextView` in un fenestra.

```
addTextView name scrollBars n = do
  sw <- scrolledWindowNew Nothing Nothing
  set sw [
    scrolledWindowVscrollbarPolicy :=
      if scrollBars then PolicyAlways else PolicyNever,
    scrolledWindowHscrollbarPolicy := PolicyAutomatic ]
  view <- textViewNew
  widgetSetName view name
  widgetAddEvents view [PointerMotionMask,LeaveNotifyMask]
  buffer <- textViewGetBuffer view
  table <- textBufferGetTagTable buffer
  font <- fontDescriptionFromString "Monospace 9"
  widgetModifyFont view (Just font)
  widgetModifyBase view StateNormal (gray 0.94)
  ct <- widgetGetPangoContext view
  t <- layoutText ct "^_^"
  (_,Rectangle x1 y1 x2 y2) <- layoutGetPixelExtents t
  widgetSetSizeRequest sw (-1) (n*(y2-y1))
  containerAdd (toContainer sw) view
  return (sw,(view,buffer,table))
```

Nos crea un fenestro con tres componentes de typo `TextView`.

```
createWindow = do
  window <- windowNew
  vbox1 <- vBoxNew False 0
  containerAdd window vbox1
```

```
  (sw1,vbf1) <- addTextView "view1" True 5
  (sw2,vbf2) <- addTextView "view2" False 1
  (sw3,vbf3) <- addTextView "view3" False 3
  set window [
    windowDefaultWidth := 480,
    windowDefaultHeight := 300
    ]
  boxPackStart vbox1 sw1 PackGrow 0
  boxPackStart vbox1 sw2 PackNatural 1
  boxPackStart vbox1 sw3 PackNatural 0
  return (window,[vbf1,vbf2,vbf3])
```

Responder a eventos

Nos responde a eventos movimento, partir, eliger, cambiar e quitar.

```
view `on` motionNotifyEvent $ vwMotion var dictTree regulas vbfs
view `on` leaveNotifyEvent $ vwLeave var
view `on` buttonPressEvent $ vwButtonPress var view buffer
timeoutAdd (timeoutFunc var vbfs dictTree regulas) 50

buffer `on` bufferChanged $ bfChanged var buffer
window `on` objectDestroy $ mainQuit
```

Trovar le parola sub cursor

Nos sape le coordinates de evento e pote calcular le position in texto per componente `Iter`. Le componente `Iter` anque da nos methodos pro trovar le initio e fino de parola.

```
wrdUnderCursor var view buffer mouseX mouseY log = do
  (x,y) <- textViewWindowToBufferCoords view TextWindowText
      (round mouseX, round mouseY)
  (iter, _) <- textViewGetIterAtPosition view x y
  start <- textIterCopy iter
  end <- textIterCopy iter
```

```
  startsWord <- textIterStartsWord start
  endsWord <- textIterEndsWord end
  insideWord <- textIterInsideWord iter
  when (log > 0) (print ("startsWord = ",startsWord,
    "endsWord = ",endsWord,"insideWord = ",insideWord))
  when (insideWord && not startsWord)
    (void $ textIterBackwardWordStart start)
  when (insideWord && not endsWord)
    (void $ textIterForwardWordEnd end)
  value <- textBufferGetText buffer start end True
  (old,o,wd) <- readMVar var
  modifyMVar_ var (\_ ->
    return (0,MouseMoved,glibToString value))
  return value
```

Nostre programma cognosce un grosso de parolas (Figura 5).

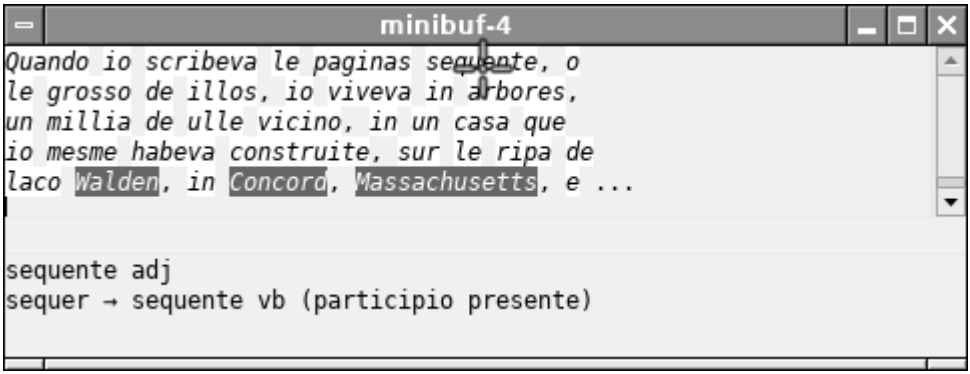

Figura 5. Nostre programma cognosce un grosso de parolas.

Ecce le programma.

```
import Graphics.UI.Gtk
import qualified Graphics.UI.Gtk.Gdk.EventM as M
import Graphics.UI.Gtk.ModelView as Model
import Control.Monad.Trans
import Control.Monad
import Control.Concurrent.MVar
import Control.Exception
```

```
import Text.Regex.Posix
import Text.Regex
import Text.Regex.Base
import Data.Array
import System.Glib.UTFString
import qualified Data.Map as Map
import Data.Maybe
import Data.Char
import Data.List.Split
import Data.List

data Changed = TextChanged | MouseMoved | Painted | Leaved
  deriving (Eq,Show)

re = "[[:alpha:]]+"
reX = makeRegex re :: Regex

matchesA2 value = map elems $ matchAll reX (value :: String)
matchesA3 value = map assocs $ matchAll reX (value :: String)
matchesA4 value = map (!0) $ matchAll reX (value :: String)

subsK value = subRegex reX value "*\\0*"

gray n = Color gt gt gt
  where
    gt = round (n * 65535)

prolongar n x = x ++ replicate r ' '
  where
    l = length x
    r = n - l

prolongar2 n x y = x ++ replicate r ' ' ++ y
  where
    l = length x + length y
    r = n - l
```

```
grays = map gray [1.0,0.9..0.0]

createTag table n = do
  tag <- textTagNew (Just (stringToGlib ("gray" ++ show n)))
  let fg = if n > length grays `div` 2
             then head grays else last grays
  set tag [
    textTagStyle := StyleItalic,
    textTagForegroundGdk := fg,
    textTagBackgroundGdk := grays !! n]
  textTagTableAdd table tag

createTags table = do
 mapM (createTag table) [0 .. length grays - 1]

textBufferGetValue buffer = do
  start <- textBufferGetStartIter buffer
  end <- textBufferGetEndIter buffer
  value <- textBufferGetText buffer start end True
  return value

paintTag buffer ((a,b),n) = do
  iter0 <- textBufferGetIterAtOffset buffer a
  iter1 <- textBufferGetIterAtOffset buffer (a+b)
  textBufferApplyTagByName buffer
    (stringToGlib ("gray" ++ show n)) iter0 iter1

paintTag3 buffer ((a,b),name,member)  = do
  let n = if member then 0 else 6
  paintTag buffer ((a,b),n)

paintTags2 buffer xs = do
  mapM_ (paintTag buffer) (zip xs [0 .. length grays - 1])

slice xs (start,n) = take n (drop start xs)

tabula1 xs = res1
```

```
  where
    res1 = [prolongar2 n a b | (a,b) <- xs]
    n = maximum [length a + length b + 1 | (a,b) <- xs]

tabula2 ws ks = map (intercalate " ") res2
  where
    res2 = transpose cs
    cs = map tabula1 rows
    rows = chunksOf c xs
    c = l `div` cols + if (l `mod` cols > 0) then 1 else 0
    l = length xs
    xs = zip ws ms
    ms = map show ks
    cols = 4

bfChanged var buffer = do
  (old,o,wd) <- readMVar var
  insMark <- textBufferGetInsert buffer
  insIter <- textBufferGetIterAtMark buffer insMark
  offset <- textIterGetOffset insIter
  modifyMVar_ var (\_ -> return (0,TextChanged,wd))

matchingWds tree rules wd =
  [ (w,x) | (w,x) <- wrs, w `Map.member` tree  ]
  where
    wd1 = map toLower wd
    vrs = ruleEtExpls wd1 rules
    wrs = [(wd1,w2)] ++ vrs
    wr =  Map.lookup wd1 tree
    w1 = fromMaybe [] wr
    w2 = intercalate " " w1

isMember tree rules wd = or [ w `Map.member` tree | w <- ws ]
  where
    variants = applyRules2 wd rules
    ws = [wd] ++ variants
```

```
paintBuffer1 buffer dictTree rules = do
  start <- textBufferGetStartIter buffer
  end <- textBufferGetEndIter buffer
  textBufferRemoveAllTags buffer start end
  value <- textBufferGetValue buffer
  let
    xs = matchesA2 value
    ts = matchesA4 value
    ws1 = map (slice value) ts
    ws = map (map toLower) ws1
    ks = map (isMember dictTree rules) ws
    ys = zip3 ts ws ks
    table = tabula2 ws ks
  mapM_ (paintTag3 buffer) ys

paintBuffer var buffer tree rules = do
  (old,o,wd) <- readMVar var
  modifyMVar_ var (\_ -> return (0,Painted,wd))
  paintBuffer1 buffer tree rules
  return ()

showT (x,y) = x ++ " " ++ y

showL xs = intercalate "\n" (map showT xs)

paintOtherWin var vbfs tree rules = do
  (old,o,wd) <- readMVar var
  let
    (view1,buffer1,table1) = vbfs !! 0
    (view3,buffer3,table3) = vbfs !! 2
    f1 = matchingWds tree rules wd
    txt = showL f1
  liftIO $ textBufferSetText buffer3 txt
  liftIO $ putStrLn txt
  modifyMVar_ var (\_ -> return (0,Painted,wd))
  return ()
```

```
timeoutFunc var vbfs tree rules = do
  let
    (view1,buffer1,table1) = vbfs !! 0
    (view3,buffer3,table3) = vbfs !! 2
  (old,o,wd) <- readMVar var
  modifyMVar_ var (\_ -> return (old+1,o,wd))
  when (old >= 9 && o == TextChanged)
    (paintBuffer var buffer1 tree rules)
  when (old >= 9 && o == MouseMoved)
    (paintOtherWin var vbfs tree rules)
  return True

breakTab w = (map toLower a, words (drop 1 b))
  where
    (a,b) = break (=='\t') w

isWord w = w == takeWhile isAlpha w

ruleEtExpl [r,repl,expl] w =
  if res1 /= w then (res1, expl2) else ("","")
  where
    rplus = "^" ++ r ++ "$"
    reX = makeRegex rplus :: Regex
    res1 = subRegex reX w repl
    expl2 = "→ " ++ w ++ " "++ expl
ruleEtExpl _ _ = ("","")

ruleEtExpls w rs = filter (/= ("","")) [ruleEtExpl r w | r <- rs]

applyRule [r,repl,expl] w = if result /= w then result else ""
  where
    rplus = "^" ++ r ++ "$"
    reX = makeRegex rplus :: Regex
    result = subRegex reX w repl
applyRule _ _ = ""

applyRules2 w rs = filter (not . null) [applyRule r w | r <- rs]
```

```
createRules = map (splitOn "\t") . splitOn "\n"

createWs2 dictText = ds3
  where
    ls = lines dictText
    ds1 = [breakTab w | w <- ls]
    ds2 = filter (\(a,b) -> isWord a) ds1
    ds3 = Map.fromList ds2

put1 (a,b) = do
  putStr a
  print b
  putStrLn ""

wrdUnderCursor var view buffer mouseX mouseY log = do
  (x,y) <- textViewWindowToBufferCoords view TextWindowText
      (round mouseX, round mouseY)
  (iter, _) <- textViewGetIterAtPosition view x y
  start <- textIterCopy iter
  end <- textIterCopy iter
  startsWord <- textIterStartsWord start
  endsWord <- textIterEndsWord end
  insideWord <- textIterInsideWord iter
  when (log > 0) (print ("startsWord = ",startsWord,
    "endsWord = ",endsWord,"insideWord = ",insideWord))
  when (insideWord && not startsWord)
    (void $ textIterBackwardWordStart start)
  when (insideWord && not endsWord)
    (void $ textIterForwardWordEnd end)
  value <- textBufferGetText buffer start end True
  (old,o,wd) <- readMVar var
  modifyMVar_ var (\_ ->
    return (0,MouseMoved,glibToString value))
  return value

setMouse view = do
```

```
  drawWindowM <- textViewGetWindow view TextWindowText
  display <- displayGetDefault
  cursor2 <- cursorNewFromName (fromJust display) "crosshair"
  let cursor = fromJust cursor2
      drawWindow = fromJust drawWindowM
  drawWindowSetCursor drawWindow (Just cursor)

vwLeave1 var = do
  (old,o,wd) <- readMVar var
  modifyMVar_ var (\_ -> return (0,Leaved,wd))

vwLeave var = do
  liftIO $ vwLeave1 var
  return False

vwMotion var tree rules vbfs = do
  let
    (view1,buffer1,table1) = vbfs !! 0
    (view3,buffer3,table3) = vbfs !! 2
  liftIO $ setMouse view1
  (mouseX,mouseY) <- M.eventCoordinates
  liftIO $ wrdUnderCursor var view1 buffer1 mouseX mouseY 0
  return False

vwButtonPress var view buffer  = do
  (mouseX,mouseY) <- M.eventCoordinates
  value <- liftIO $ wrdUnderCursor var view buffer mouseX mouseY 1
  return False

addTextView name scrollBars n = do
  sw <- scrolledWindowNew Nothing Nothing
  set sw [
    scrolledWindowVscrollbarPolicy :=
      if scrollBars then PolicyAlways else PolicyNever,
    scrolledWindowHscrollbarPolicy := PolicyAutomatic ]
  view <- textViewNew
  widgetSetName view name
```

```
  widgetAddEvents view [PointerMotionMask,LeaveNotifyMask]
  buffer <- textViewGetBuffer view
  table <- textBufferGetTagTable buffer
  font <- fontDescriptionFromString "Monospace 9"
  widgetModifyFont view (Just font)
  widgetModifyBase view StateNormal (gray 0.94)
  ct <- widgetGetPangoContext view
  t <- layoutText ct "^_^"
  (_,Rectangle x1 y1 x2 y2) <- layoutGetPixelExtents t
  widgetSetSizeRequest sw (-1) (n*(y2-y1))
  containerAdd (toContainer sw) view
  return (sw,(view,buffer,table))

createWindow = do
  window <- windowNew
  vbox1 <- vBoxNew False 0
  containerAdd window vbox1
  (sw1,vbf1) <- addTextView "view1" True 5
  (sw2,vbf2) <- addTextView "view2" False 1
  (sw3,vbf3) <- addTextView "view3" False 3
  set window [
    windowDefaultWidth := 480,
    windowDefaultHeight := 300
    ]
  boxPackStart vbox1 sw1 PackGrow 0
  boxPackStart vbox1 sw2 PackNatural 1
  boxPackStart vbox1 sw3 PackNatural 0
  return (window,[vbf1,vbf2,vbf3])

main = do
  initGUI
  var <- newMVar (9,TextChanged,"")
  content <- readFile "thoreau-initio-2.txt"
  dictText <- readFile "51500-tab.txt"
  rulesText <- readFile "regulas-5.txt"
  (window,vbfs) <- createWindow
  let
```

```
    dictTree = createWs2 dictText
    regulas = createRules rulesText
    (view,buffer,table) = vbfs !! 0
  textBufferSetText buffer content
  view `on` motionNotifyEvent $ vwMotion var dictTree regulas vbfs
  view `on` leaveNotifyEvent $ vwLeave var
  view `on` buttonPressEvent $ vwButtonPress var view buffer
  timeoutAdd (timeoutFunc var vbfs dictTree regulas) 50

  buffer `on` bufferChanged $ bfChanged var buffer
  window `on` objectDestroy $ mainQuit

  widgetShowAll window
  createTags table
  mainGUI
```

Libreria `Graphics.UI.Gtk.Abstract.Widget`

Ulle proprietates del `TextView` es in su subclasse `Widget`.

Typos:

```
GType  KeyVal StockId   AccelFlags Requisition  WidgetHelpType
Color  Region EventMask Allocation ExtensionMode DirectionType
Widget Bitmap Rectangle  StateType TextDirection
```

Prefixo: `widget-`
Methodos:

```
~Show     ~SizeAllocate ~SetDirection ~InputShapeCombineMask
~Hide     ~SetAccelPath ~GetDirection  ~TranslateCoordinates
~Path     ~SetSensitive ~CreateLayout  ~SetScrollAdjustments
~ShowNow  ~PushColormap ~GetClipboard   ~GetChildRequisition
~ShowAll   ~QueueResize ~SetNoShowAll   ~QueueResizeNoRedraw
~HideAll   ~GrabDefault ~GetNoShowAll   ~SetDefaultDirection
~Destroy   ~GetToplevel ~IsComposited   ~GetDefaultDirection
~SetName   ~GetAncestor ~KeynavFailed   ~SetRedrawOnAllocate
~GetName   ~GetColormap ~GetHasWindow   ~RemoveMnemonicLabel
~HasGrab   ~SetColormap ~SetHasWindow   ~TriggerTooltipQuery
~GetSize   ~SizeRequest ~GetSensitive    ~SetExtensionEvents
~Activate  ~PopColormap ~GetDrawWindow   ~GetExtensionEvents
~SetStyle  ~GetSnapshot ~QueueDrawArea   ~SetDefaultColormap
~GetStyle  ~ModifyStyle ~GetAccessible   ~GetDefaultColormap
~ModifyFg  ~RestoreText ~GetRootWindow   ~CreatePangoContext
```

```
~ModifyBg  ~RestoreBase ~GetHasTooltip   ~ListMnemonicLabels
~Reparent  ~ResetShapes ~SetHasTooltip   ~SetReceivesDefault
~GetState  ~GetSettings ~GetAllocation   ~GetReceivesDefault
~SetState  ~GetCanFocus ~GetCanDefault    ~RemoveAccelerator
~QueueDraw ~SetCanFocus ~SetCanDefault    ~SetDoubleBuffered
~Intersect ~IsSensitive ~GetHasDefault     ~CanActivateAccel
~GrabFocus ~GetHasFocus ~GetSavedState     ~ShapeCombineMask
~DelEvents  ~GetIsFocus ~AddAccelerator    ~GetCompositeName
~AddEvents  ~GetPointer ~SetSensitivity    ~GetModifierStyle
~GetEvents  ~IsAncestor ~GetSizeRequest    ~SetCompositeName
~SetEvents  ~ModifyText ~SetSizeRequest    ~MnemonicActivate
~ClassPath  ~ModifyBase ~GetTooltipText    ~AddMnemonicLabel
~RestoreFg  ~ModifyFont ~SetTooltipText    ~GetTooltipMarkup
~RestoreBg  ~RenderIcon ~HasIntersection   ~SetTooltipMarkup
~GetParent  ~ChildFocus ~GetParentWindow   ~GetTooltipWindow
~GetScreen  ~GetDisplay ~GetDefaultStyle   ~SetTooltipWindow
~HasScreen  ~GetVisible ~GetPangoContext    ~GetChildVisible
~GetAction  ~IsDrawable ~SetAppPaintable    ~SetChildVisible
~ErrorBell  ~IsToplevel ~RegionIntersect    ~GetAppPaintable
~GetWindow   ~SetWindow
```

Attributos:

```
~Name   ~CanFocus ~MarginTop ~MarginRight ~TooltipText ~ReceivesDefault
~Style  ~HasFocus ~Sensitive  ~CanDefault ~GetRealized ~ExtensionEvents
~State  ~Colormap ~NoShowAll  ~HasDefault ~SetRealized  ~CompositeChild
~Parent  ~Opacity ~Direction  ~HExpandSet ~WidthRequest  ~HeightRequest
~Events  ~IsFocus ~GetMapped  ~VExpandSet ~MarginBottom  ~CompositeName
~Expand  ~HExpand ~SetMapped  ~HasTooltip ~AppPaintable  ~TooltipMarkup
~Visible ~VExpand ~MarginLeft ~HasRcStyle ~ChildVisible
```

Signales:

```
focus   showSignal hideSignal screenChanged popupMenuSignal
realize  unrealize grabNotify  sizeAllocate hierarchyChanged
styleSet mapSignal unmapSignal stateChanged directionChanged
showHelp parentSet sizeRequest queryTooltip accelClosuresChanged
```

Eventos:

```
mapEvent  grabBrokenEvent keyReleaseEvent visibilityNotifyEvent
unmapEvent configureEvent buttonPressEvent   buttonReleaseEvent
deleteEvent focusOutEvent enterNotifyEvent    motionNotifyEvent
exposeEvent keyPressEvent leaveNotifyEvent    proximityOutEvent
scrollEvent noExposeEvent proximityInEvent     windowStateEvent
destroyEvent focusInEvent
```

Libreria `Graphics.UI.Gtk.Gdk.EventM`

Alicun typos e functiones nos trova in libreria `Graphics.UI.Gtk.Gdk.EventM`.

Typos:

```
EAny  EButton EMotion ECrossing EConfigure EVisibility EWindowState
```

```
EKey  EScroll EExpose EProperty EProximity EGrabBroken EOwnerChange
EventM EFocus
```

Prefixo: `event-`
Functiones:

```
~Sent  ~Modifier ~Implicit  ~KeyboardGroup ~SelectionTime
~Time  ~Position stopEvent  ~CrossingFocus ~RootCoordinates
~Area   ~KeyName ~Selection  ~CrossingMode ~HardwareKeycode
~Size   ~FocusIn ~NotifyType ~ChangeReason ~ScrollDirection
~Window tryEvent ~GrabWindow ~KeyboardGrab ~VisibilityState
~KeyVal  ~IsHint currentTime  ~ModifierAll ~WindowStateChanged
~Button  ~Region ~Coordinates ~WindowState
```