Luku 1

Gtk-käyttöliittymä

Tässä luvussa tutustumme Gtk-käyttöliittymän kirjoittamiseen. Gtk (*the Gimp toolkit*) on suosittu käyttöliittymäkirjasto, joka aikoinaan syntyi Gimp-kuvankäsittelyohjelman komponenttikirjastona, ja joka nykyisin toimii useimpien Linux-koneiden työpöytäympäristön perusrakennusosana.

1.1 Ohjelmaikkuna, jossa yksinkertainen painike

Ensimmäisessä esimerkkiohjelmassamme luomme ikkunan, joka sisältää yksinkertaisen painikkeen tekstillä ”Click me!” (kuva 1).

Kuva 1. Yksinkertainen painike.

```
import Graphics.UI.Gtk
import Control.Monad.Trans (liftIO)

main = do
  initGUI
  window <- windowNew
```

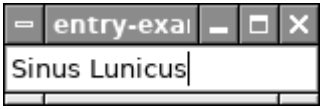
```
  button <- buttonNewWithLabel "Click me!"
  containerAdd window button
  widgetShowAll window
  button `on` buttonPressEvent $ tryEvent $ whenClicked
  onDestroy window mainQuit
  mainGUI

whenClicked = do
  liftIO $ putStrLn "Button was clicked."
```

Luomme ikkunan komennolla `windowNew` ja painikkeen komennolla `buttonNewWithLabel`. Liitämme painikkeen ikkunaan komennolla `containerAdd`. Esitämme ikkunan ja kaikki sen sisältämät alikomponentit komennolla `widgetShowAll`. Asetamme painikkeelle tapahtumankäsittelijän `whenClicked`. Ohjelma kutsuu tapahtumankäsittelijää, aina kun painike lähettää signaalin `buttonPressEvent` eli painiketta painettaessa. Ikkunan sulkeminen (`onDestroy`) päättää ohjelman suorituksen (`mainQuit`).

Tuomme kirjastosta `Control.Monad.Trans` funktion `liftIO`, jonka avulla voimme yhteensovittaa tapahtumankäsittelijän vaatiman tyypin `EventM` sekä syöte- ja tulostustyypin `IO`.

1.2 Ohjelmaikkuna, jossa tekstinsyöttökenttä

Lisäsimme edellä ohjelmaikkunaan painikkeen komennolla `buttonNewWithLabel`. Myös muiden komponenttien lisäys tapahtuu samaa nimeämislogiikkaa noudattaen. Esimerkiksi yksinkertaisen tekstinsyöttökentän (`Entry`) lisäämme komennolla `entryNew` (kuva 2).



Kuva 2. Tekstinsyöttökenttä.

```
import Graphics.UI.Gtk
```

```
main = do
  initGUI
  window <- windowNew
  entry <- entryNew
  containerAdd window entry
  widgetShowAll window
  onDestroy window mainQuit
  mainGUI
```

1.3 Ohjelmaikkuna puunäkymällä

Puunäkymän lisääminen on hieman monimutkaisempi toimenpide, sillä puunäkymä tarvitsee näkymän (*view*) lisäksi myös tietomallin (*model*). Tietomallin pohjana voi olla yksinkertainen lista. Muodostamme listasta tietomallin komennolla `listStoreNew`.

Kun haluamme lisätä ikkunaan useampia komponentteja, on meidän päätettävä komponenttien asettelusta. Käytämme seuraavassa komponenttien asettelemiseksi allekkain pystysuuntaista laatikkoa `VBox` (*vertical box*). Asettelemme komponentit laatikkoon komennolla `boxPackStart` (kuva 3).

Kuva 3. Tekstinsyöttökenttä ja puunäkymä.

```
import Graphics.UI.Gtk
import Graphics.UI.Gtk.ModelView as Model
```

```
main = do
  initGUI
  window <- windowNew
  vbox1 <- vBoxNew False 0
  entry <- entryNew
  containerAdd window vbox1
  list <- listStoreNew [
    "Lacus Somniorum", "Lacus Doloris", "Lacus Timoris",
    "Palus Putredinis", "Lacus Autumni"]
  treeview <- Model.treeViewNewWithModel list
  Model.treeViewSetHeadersVisible treeview False
  col <- Model.treeViewColumnNew
  renderer <- Model.cellRendererTextNew
  Model.cellLayoutPackStart col renderer False
  Model.cellLayoutSetAttributes col renderer list
    (\text -> [Model.cellText := text])
  Model.treeViewAppendColumn treeview col
  tree <- Model.treeViewGetSelection treeview
  boxPackStart vbox1 entry PackRepel 0
  boxPackStart vbox1 treeview PackRepel 0
  widgetShowAll window
  onDestroy window mainQuit
  mainGUI
```

1.4 Tuloslistan suodattaminen säännöllisillä lausekkeilla

Kirjoitamme seuraavaksi ohjelman, joka suodattaa tuloslistan tekstikentässä antamamme säännöllisen lausekkeen avulla.

Luemme ensin komennolla `readFile` tekstitiedoston, josta suodatamme pois tyhjät rivit.

Määrittelemme näppäimen vapautuksen (`keyReleaseEvent`) tapahtumankäsittelijäksi funktion `updateList1`, joka lukee tekstikentän sisällön, kun sitä on muutettu.

Säännöllisten lausekkeiden käsittelyyn tuomme kirjaston `Text.Regex.Posix`, joka tarjoaa funktion `(=~)`. Funktio `(=~)` on monimuotoinen ja vaatii siksi yleisessä muodossaan tyyppimäärittelyn. Kun valitsemme lausekkeen `a =~ b` tyypiksi `Bool`, palauttaa lauseke arvon `True`, mikäli säännöllinen lauseke `b` esiintyy merkkijonossa `a`. Mikäli valitsemme lausekkeen `a =~ b` tyypiksi `String`, lauseke palauttaa ensimmäisen osuman. Listauksessamme käytämme funktiota `(=~)` funktion `filter` ensimmäisenä argumenttina, joten kääntäjä päättelee tyypin olevan `Bool`.

```
> import Text.Regex.Posix
> "abcd" =~ "a" :: Bool
True
> "abcd" =~ "a" :: String
"a"
```

Kuvassa 4 olemme syöttäneet tekstikenttään säännöllisen lausekkeen `(.)\1`. Säännöllisissä lausekkeissa piste `.` vastaa mitä tahansa merkkiä. Sulkumerkit `()` muodostavat ensimmäisen alilausekkeen. Merkintä `\1` vastaa ensimmäistä alilauseketta. Kokonaisuudessaan säännöllinen lauseke `(.)\1` vastaa siten kahta peräkkäistä samaa merkkiä. Esimerkissämme tekstirivejä, joilla esiintyy kaksi peräkkäistä samaa merkkiä, löytyy yhteensä 10 kappaletta.

1.5 Poikkeuksien käsittely

Kun säännöllinen lauseke ei ole kelvollinen, nostaa järjestelmä poikkeuksen (*exception*). Esimerkiksi keskeneräiset lausekkeet kuten "`(.`" eivät ole kelvollisia. Poikkeuksien käsittelyyn tuomme kirjaston `Control.Exception`, joka tarjoaa muun muassa funktiot `try` ja `evaluate`. Näistä `try` on monimuotoinen funktio ja vaatii siksi yleisessä muodossaan tyyppimäärittelyn. Annamme tyyppimäärittelyn funktion `tryFilter` tyyppiallekirjoituksessa.

```
tryFilter :: String -> [String]
          -> IO (Either SomeException [String])
tryFilter txt list = do
  result <- try (evaluate (filter (=~ txt) list))
  return result
```

Kuva 4. Tuloslista suodatettuna säännöllisellä lausekkeella `(.)\1`.

Tyypin `Either` mahdolliset arvot ovat `Left x` ja `Right y`. Funktio `try` palauttaa arvon `Left x` silloin, kun argumenttina antamamme funktion suoritus on keskeytynyt poikkeukseen `x` sekä arvon `Right y` silloin, kun antamamme funktio onnistuneesti palauttaa arvon `y`. Esimerkissämme samaistamme virheellisen säännöllisen lausekkeen lausekkeeseen, jonka tulosjoukko on tyhjä lista `[]`.

```
filter1 txt list = do
  result <- tryFilter txt list
  return $ case result of
    Left ex -> []
    Right val -> val
```

Esitämme seuraavassa ohjelmakoodin kokonaisuudessaan.

```
import Graphics.UI.Gtk
import Graphics.UI.Gtk.ModelView as Model
import Control.Monad.Trans (liftIO)
import Text.Regex.Posix
import Control.Exception
```

```
main = do
  content <- readFile "moon-latin.txt"
  let moon = filter (not . null) (lines content)
  initGUI
  window <- windowNew
  vbox1 <- vBoxNew False 0
  entry <- entryNew
  containerAdd window vbox1
  listore <- listStoreNew []
  treeview <- Model.treeViewNewWithModel listore
  entry `on` keyReleaseEvent $ do
    liftIO $ updateList1 entry listore moon
  Model.treeViewSetHeadersVisible treeview False
  col <- Model.treeViewColumnNew
  renderer <- Model.cellRendererTextNew
  Model.cellLayoutPackStart col renderer False
  Model.cellLayoutSetAttributes col renderer listore
    (\text -> [Model.cellText := text])
  Model.treeViewAppendColumn treeview col
  tree <- Model.treeViewGetSelection treeview
  boxPackStart vbox1 entry PackNatural 0
  boxPackStart vbox1 treeview PackNatural 0
  widgetShowAll window
  liftIO $ updateList1 entry listore moon
  onDestroy window mainQuit
  mainGUI

tryFilter :: String -> [String]
          -> IO (Either SomeException [String])
tryFilter txt list = do
  result <- try (evaluate (filter (=~ txt) list))
  return result

filter1 txt list = do
  result <- tryFilter txt list
  return $ case result of
```

```
    Left ex -> []
    Right val -> val

updateList1 entry listore list = do
  listStoreClear listore
  txt <- entryGetText entry
  list1 <- listStoreToList listore
  list2 <- filter1 txt list
  let
    n = 8
    list3 = take (n + 1) list2
    l2 = length list2
    l3 = length list3
    list4
      | l2 == l3 = take (n + 1) (list2 ++ repeat "")
      | l2 > l3  = take n list3 ++
        ["(+" ++ show (l2 - l3 + 1) ++ " others)"]
  mapM_ (listStoreAppend listore) list4
  return True
```

1.6 Ohjelma Png-kuvan näyttämiseen

Seuraavassa esimerkkiohjelmassa emme käytä moniakaan Gtk-kirjaston visuaalisia komponentteja vaan kirjoitamme lyhyen yleiskäyttöisen ohjelman Png-muotoisen kuvan esittämiseksi ruudulla.

```
import Control.Concurrent.MVar
import System.IO.Unsafe
import System.Environment (getArgs)
import Graphics.UI.Gtk
import qualified Graphics.Rendering.Cairo as C
import qualified Graphics.UI.Gtk.Gdk.EventM as M
import System.Glib.UTFString (glibToString)

firstArg args =
  case args of
```

```
    [] -> error "must supply a file to open"
    [arg] -> arg
    _ -> error "too many arguments"

main = do
  args <- getArgs
  let arg1 = firstArg args
  initGUI
  var <- newMVar (1.0,0.0,0.0)
  vPos <- newMVar (None,0.0,0.0)
  window <- windowNew
  canvas <- drawingAreaNew
  surf <- return $ unsafeLoadPNG arg1
  widgetAddEvents canvas [Button1MotionMask]
  widgetSetSizeRequest canvas 300 200
  centerImg var surf canvas
  canvas `on` motionNotifyEvent $ do
    (mouseX,mouseY) <- eventCoordinates
    t <- M.eventTime
    C.liftIO $
      changePos vPos var surf canvas mouseX mouseY
    C.liftIO $ logMsg 0 ("Motion Time: " ++ s t)
    return False
  window `on` keyPressEvent $ tryEvent $ do
    key <- eventKeyName
    keyInput var surf canvas (glibToString key)
    C.liftIO $ updateCanvas1 var canvas surf
    return ()
  canvas `on` buttonPressEvent $ tryEvent $ do
    (mouseX,mouseY) <- printMouse
    C.liftIO $ printPointer canvas
    C.liftIO $ printMVar var mouseX mouseY
    C.liftIO $ modifyMVar_ vPos (\_ ->
      return (Press,mouseX,mouseY))
  canvas `on` buttonReleaseEvent $ tryEvent $ do
    (mouseX,mouseY) <- M.eventCoordinates
    m <- M.eventModifier
```

```
    b <- M.eventButton
    (cause,vPosX,vPosY) <- C.liftIO $ readMVar vPos
    C.liftIO $ release cause b var vPosX vPosY
  canvas `on` scrollEvent $ tryEvent $ do
    (mouseX,mouseY) <- M.eventCoordinates
    m <- M.eventModifier
    d <- M.eventScrollDirection
    t <- M.eventTime
    C.liftIO $ changeRef var d mouseX mouseY
    C.liftIO $ updateCanvas1 var canvas surf
    C.liftIO $ logMsg 0 ("Scroll: " ++ s t ++ s mouseX ++
      s mouseY ++ s m ++ s d)
  onDestroy window mainQuit
  onExpose canvas $ const (updateCanvas1 var canvas surf)
  set window [containerChild := canvas]
  widgetShowAll window
  mainGUI

data EvtType = Press | Release | Move | Scroll | None

release Press button var mouseX mouseY = do
  (varS,varX,varY) <- readMVar var
  let
    x = (mouseX - varX) / varS
    y = (mouseY - varY) / varS
  C.liftIO $ logMsg 0
    ("Add point: " ++ s x ++ s y ++ s button)
  C.liftIO $ logMsg 1 (s x ++ s y)

release _ button var x y = do
  C.liftIO $ logMsg 0 ("Release (other): " ++ s x ++ s y)

changePos vPos var surf canvas mouseX mouseY = do
  (cause,vPosX,vPosY) <- readMVar vPos
  (scaleOld,oldX,oldY) <- readMVar var
  let
    dx = vPosX - mouseX
```

```
    dy = vPosY - mouseY
  modifyMVar_ var (\_ ->
    return (scaleOld,oldX - dx,oldY - dy))
  modifyMVar_ vPos (\_ ->
    return (Move,mouseX,mouseY))
  updateCanvas1 var canvas surf

intToDouble :: Int -> Double
intToDouble i = fromRational (toRational i)

s x = show x ++ " "

printMouse = do
  (mouseX,mouseY) <- M.eventCoordinates
  C.liftIO $ logMsg 0 ("Mouse: " ++ s mouseX ++ s mouseY)
  return  (mouseX,mouseY)

printPointer canvas = do
  (widX,widY) <- widgetGetPointer canvas
  logMsg 0 ("Widget: " ++ s widX ++ s widY)

printMVar var mouseX mouseY = do
  (varS,varX,varY) <- readMVar var
  let
    x = (mouseX - varX) / varS
    y = (mouseY - varY) / varS
  logMsg 0 ("MVar: " ++ s varS ++ s varX ++ s varY)
  logMsg 0 ("Calc: " ++ s x ++ s y)

centerImg var surf canvas = do
  w1 <- C.imageSurfaceGetWidth surf
  h1 <- C.imageSurfaceGetHeight surf
  (w2,h2) <- widgetGetSizeRequest canvas
  let
    dh = intToDouble (h2 - h1)
    dw = intToDouble (w2 - w1)
  modifyMVar_ var (\_ -> return (1.0,dw / 2,dh / 2))
```

```
keyInput var surf canvas key = do
  C.liftIO $ print key
  case key of
    "q" -> do
      C.liftIO $ mainQuit
    "1" -> do
      C.liftIO $ centerImg var surf canvas

changeRef var d mouseX mouseY = do
  (scaleOld,oldX,oldY) <- readMVar var
  let
    scaleD = scale1 d
    scaleNew = scaleD * scaleOld
    dx = (mouseX - oldX) * (scaleD - 1)
    dy = (mouseY - oldY) * (scaleD - 1)
    newX = oldX - dx
    newY = oldY - dy
    result =  (scaleNew,newX,newY)
  modifyMVar_ var (\_ -> return result)
  logMsg 0 ("Change MVar: " ++ s scaleNew ++
    s newX ++ s newY)
  where
    factor = 5 / 4
    scale1 ScrollUp   = factor
    scale1 ScrollDown = 1 / factor

updateCanvas1 var canvas surf = do
  win <- widgetGetDrawWindow canvas
  (width, height) <- widgetGetSize canvas
  renderWithDrawable win $
    paintImage1 var surf
  return True

imageSurfaceCreateFromPNG :: FilePath -> IO C.Surface
imageSurfaceCreateFromPNG file =
  C.withImageSurfaceFromPNG file $ \png -> do
```

```
    C.liftIO $ logMsg 0 "Load Image"
    w <- C.renderWith png $ C.imageSurfaceGetWidth png
    h <- C.renderWith png $ C.imageSurfaceGetHeight png
    surf <- C.createImageSurface C.FormatRGB24 w h
    C.renderWith surf $ do
      C.setSourceSurface png 0 0
      C.paint
    return surf

unsafeLoadPNG file = unsafePerformIO $
  imageSurfaceCreateFromPNG file

paintImage1 var surf = do
  (sc,x,y) <- C.liftIO $ readMVar var
  C.setSourceRGB 1 1 1
  C.paint
  C.translate x y
  C.scale sc sc
  C.liftIO $ logMsg 0 ("Paint Image: " ++
    s sc ++ s x ++ s y)
  C.setSourceSurface surf 0 0
  C.paint

logMsg 0 s = do
  return ()
logMsg 1 s = do
  putStrLn s
  return ()
```

1.7 Piirtoalue `DrawingArea`

Luomme ikkunaan piirtoalueen komennolla `drawingAreaNew`. Annamme piirtoalueelle nimen `canvas`. Piirtoalueen kokopyynnön asetamme komennolla `widgetSetSizeRequest`.

Luemme piirtoalueelle kuvan määrittelemällä komennon `imageSurfaceCre-`

`ateFromPNG`. Kuvan keskittämiseksi piirtoalueelle määrittelemme komennon `centerImg`.

Luomme tapahtumankäsittelijöiden avulla käyttäjälle mahdollisuuden siirtää kuvaa ja muuttaa kuvan kokoa. Kun kuvaa on siirretty tai sen kokoa muutettu, piirrämme kuvan uudestaan käyttäen määrittelemäämme komentoa `updateCanvas1`. Tämä komento kutsuu piirtotyön suorittavaa rutiinia `paintImage1`, jossa kutsumme piirtokirjasto Cairon tarjoamia piirtokomentoja. Tuomme piirtokirjaston nimettynä (`qualified ... as C`), joten kaikki sen tarjoamat komennot ohjelmalistauksessa alkavat etuliitteellä "`C.`".

1.8 Tapahtumankäsittelijät

Pääohjelmassa olemme määritelleet tapahtumankäsittelijän hiiren liikkeelle (`motionNotifyEvent`), näppäimen painallukselle (`keyPressEvent`), hiiren näppäimen painallukselle (`buttonPressEvent`), hiiren näppäimen vapauttamiselle (`buttonReleaseEvent`), hiiren rullan pyörittämiselle (`scrollEvent`), ikkunan sulkupainikkeen painamiselle (`onDestroy`) sekä piirtoalueen uudelleenpiirtämiselle (`onExpose`).

Osa tapahtumankäsittelijöistämme on hyvin yksinkertaisia, ja ne kirjoittavat ainoastaan viestin komentoikkunaan funktiokutsulla `logMsg 1` tai jättävät kirjoittamatta funktiokutsulla `logMsg 0`.

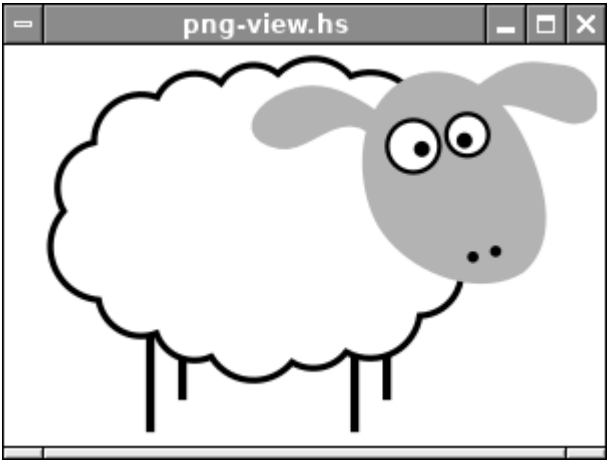
Tavallisesti hiiren liikuttaminen piirtoalueella ei tuota tapahtumasignaalia, mutta voimme halutessamme tuottaa sellaisia esimerkiksi silloin, kun hiiren ykköspainike on painettuna. Teemme näin pääohjelmassa komennolla `widgetAddEvents`.

```
widgetAddEvents canvas [Button1MotionMask]
```

Annamme ohjelmalle nimeksi png-view ja käynnistämme ohjelman komentotulkissa. Ohjelma saa argumenttinaan kuvatiedoston nimen.

```
$ runhaskell png-view foci-5.png
```

Avautuvassa ikkunassa näemme valitsemamme kuvan (kuva 5). Voimme suurentaa ja pienentää näkymää hiiren rullalla.



Kuva 5. Ohjelmaikkuna.

1.9 `MVar`-muuttujaviittaukset

Käytämme ohjelmassamme `MVar`-muuttujaviittauksia tapahtumankäsittelijöissä. Tämä on käytännöllinen tapa välittää tietoa graafisen käyttöliittymän sisällä.

Luomme uuden muuttujan komennolla `newMVar`. Muuttujan sisältämän tiedon luemme komennolla `readMVar`. Olemassaolevaa muuttujaa muutamme komennolla `modifyMVar_`.

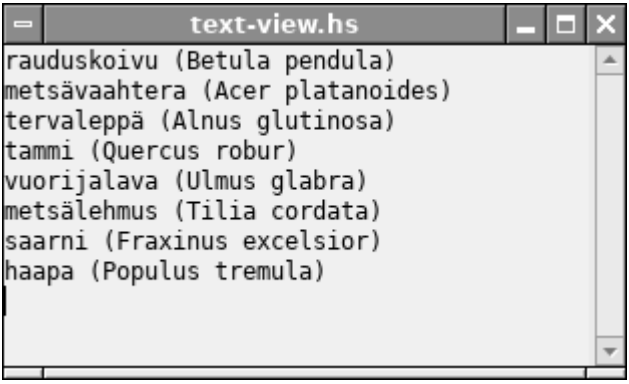
Käyttämämme muuttujaviittaukset ovat `var`, joka sisältää kuvan suurennoksen sekä vasemman ylänurkan x- ja y-koordinaatit, ja `vPos`, joka sisältää hiiritapahtuman syyn (`Press` (*painallus*), `Release` (*vapautus*), `Move` (*siirto*), `Scroll` (*rullaus*) tai `None` (*ei mikään*) sekä tapahtumahetken hiiren osoittimen x- ja y-koordinaatit.

Muuttujaviittausten sisältämän tiedon avulla ohjelma kykenee laskemaan esitettävälle kuvalle uudet koordinaatit ja suurennoksen aina tapahtumankäsittelijän sitä pyytäessä.

1.10 Tekstinäkymä ja tekstipuskuri

Gtk-kirjaston tekstinäkymä (`textView`) ja tekstipuskuri (`textBuffer`) sisältävät hyvin monipuoliset välineet yksinkertaisen tekstimuokkaimen luomiseen.

Kirjoitamme lyhyen ohjelman tekstitiedoston muokkaamiseen (kuva 6).



Kuva 6. Tekstitiedosto tekstinäkymäkomponentissa.

```
import Graphics.UI.Gtk

main = do
  initGUI
  content <- readFile "puulajit-latina.txt"
  window <- windowNew
  sw <- scrolledWindowNew Nothing Nothing
  set sw [
    scrolledWindowVscrollbarPolicy := PolicyAlways,
    scrolledWindowHscrollbarPolicy := PolicyAutomatic ]
  view <- textViewNew
  buffer <- textViewGetBuffer view
  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
  mainGUI

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

1.11 Gtk-käyttöliittymäkirjastojen hierarkia

Graphics.UI.Gtk

- Abstract
  Bin, Box, ButtonBox, Container, IMContext, Misc, Object, Paned, Range, Scale, Scrollbar, Separator, Widget.
- ActionMenuToolbar
  Action, ActionGroup, RadioAction, RecentAction, ToggleAction, UIManager.
- Builder
- Buttons
  Button, CheckButton, LinkButton, RadioButton, ScaleButton, ToggleButton, VolumeButton.
- Cairo
- Display
  AccelLabel, Image, InfoBar, Label, ProgressBar, Spinner, StatusIcon, Statusbar.
- Embedding
  Embedding, Plug, Socket, Types.
- Entry
  Editable, Entry, EntryBuffer, EntryCompletion, HScale, Spin-

Button, VScale.

Gdk

AppLaunchContext, Cursor, Display, DisplayManager, DrawWindow, Drawable, EventM, Events, GC, Gdk, Keymap, Keys, Pixbuf, PixbufAnimation, Pixmap, Region, Screen.

General

Clipboard, Drag, Enums, General, IconFactory, IconTheme, RcStyle, Selection, Settings, StockItems, Style.

Layout

Alignment, AspectFrame, Expander, Fixed, HBox, HButtonBox, HPaned, Layout, Notebook, Table, VBox, VButtonBox, VPaned.

MenuComboToolbar

CheckMenuItem, Combo, ComboBox, ComboBoxEntry, ImageMenuItem, Menu, MenuBar, MenuItem, MenuShell, MenuToolButton, OptionMenu, RadioMenuItem, RadioToolButton, SeparatorMenuItem, SeparatorToolItem, TearoffMenuItem, ToggleToolButton, ToolButton, ToolItem, ToolItemGroup, ToolPalette, Toolbar.

Misc

Accessible, Adjustment, Arrow, Calendar, DrawingArea, EventBox, HandleBox, IMContextSimple, IMMulticontext, SizeGroup, Tooltip, Tooltips, Viewport.

ModelView

CellEditable, CellLayout, CellRenderer, CellRendererAccel, CellRendererCombo, CellRendererPixbuf, CellRendererProgress, CellRendererSpin, CellRendererSpinner, CellRendererText, CellRendererToggle, CellView, CustomStore, IconView, ListStore, TreeDrag, TreeModel, TreeModelFilter, TreeModelSort, TreeRowReference, TreeSelection, TreeSortable, TreeStore, TreeView, TreeViewColumn.

Multiline

TextBuffer, TextIter, TextMark, TextTag, TextTagTable, TextView.

Ornaments

Frame, HSeparator, VSeparator.

Printing

PageSetup, PaperSize, PrintContext, PrintOperation, PrintSettings.

`Recent`
: `RecentChooser`, `RecentChooserMenu`, `RecentChooserWidget`, `RecentFilter`, `RecentInfo`, `RecentManager`.

`Scrolling`
: `HScrollbar`, `ScrolledWindow`, `VScrollbar`.

`Selectors`
: `ColorButton`, `ColorSelection`, `ColorSelectionDialog`, `FileChooser`, `FileChooserButton`, `FileChooserDialog`, `FileChooserWidget`, `FileFilter`, `FileSelection`, `FontButton`, `FontSelection`, `FontSelectionDialog`, `HSV`.

`Special`
: `HRuler`, `Ruler`, `VRuler`.

`Windows`
: `AboutDialog`, `Assistant`, `Dialog`, `Invisible`, `MessageDialog`, `OffscreenWindow`, `Window`, `WindowGroup`.

1.12 Kirjasto `Graphics.UI.Gtk.Abstract.Widget`

Useimpien komponenttien (kuten `Button`) perustyyppi on `Widget`. Kun etsimme esimerkiksi painikkeelle tapahtumankäsittelijöitä, on meidän etsittävä niitä komponentin perustyypin dokumentaatiosta.

Luettelemme seuraavassa tyypin `Widget` tarjoamat metodit, attribuutit ja tapahtumat.

Etuliite: `widget-`
Metodit:

```
~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    ~SetChildVisible
~GetAction  ~IsDrawable ~SetAppPaintable    ~GetAppPaintable
~ErrorBell  ~IsToplevel ~RegionIntersect
~GetWindow   ~SetWindow ~GetChildVisible
```

Attribuutit:

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

Signaalit:

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

Tapahtumat:

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

1.13 Kirjasto `Graphics.UI.Gtk.Gdk.EventM`

Etuliite: `event-`
Funktioita:

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