App (Class)

App.Open:

```
Sub Open()
  if not TextUtils.Init() then
    MsgBox("Init text utils fails.")
    Quit()
  end
End Sub
```

FileUtils (Module)

FileUtils.MakeSpec:

```
Protected Function MakeSpec(nr as integer,extension as string,byref spec as FolderItem) As boolea
                n
  dim a as integer
  dim b as integer
  dim c as integer
  dim d as integer
  dim a_string as string
  dim b_string as string
  dim c_string as string
  dim d_string as string
  dim s as string

  a=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  b=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  c=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  d=Bitwise.BitAnd(nr,255)

  a_string=Str(a)+extension
  b_string=Str(b)
  c_string=Str(c)
  d_string=Str(d)

  spec=new FolderItem("")
  if spec=nil then
    MsgBox("Out of memory.")
    return false
  end

  if not FileUtils.FindFolder(spec,"Output",spec) then
    MsgBox("Find folder fails.")
    return false
  end

  select case extension
  case ".jpg",".gif",".png"
    if not FileUtils.FindFolder(spec,"images",spec) then
      MsgBox("Find folder fails.")
      return false
    end
  end

  if not FileUtils.FindFolder(spec,d_string,spec) then
```

```
    MsgBox("Find folder fails.")
    return false
  end

  if not FileUtils.FindFolder(spec,c_string,spec) then
    MsgBox("Find folder fails.")
    return false
  end

  if not FileUtils.FindFolder(spec,b_string,spec) then
    MsgBox("Find folder fails.")
    return false
  end

  spec=spec.TrueChild(a_string)
  if spec=nil then
    s="File "+DQ("PageURLs.txt")+" not found."
    MsgBox(s)
    return false
  end

  return true
End Function
```

FileUtils.FindFolder:

```
Protected Function FindFolder(spec1 as FolderItem,name as string,byref spec2 as FolderItem) As b
                  oolean
  dim err as integer

  spec2=spec1.TrueChild(name)
  if spec2=nil then
    MsgBox("Out of memory.")
    return false
  end

  if not spec2.exists then
    spec2.CreateAsFolder()
    err=spec2.lastErrorCode
    if err<>0 then
      MsgBox("Create folder fails.")
      return false
    end
  end

  return true
End Function
```

FileUtils.ReadText:

```
Protected Function ReadText(spec as FolderItem,byref text as string) As boolean
  dim b as BinaryStream
  dim err as integer

  b=spec.OpenAsBinaryFile(false)
  if b.lastErrorCode<>0 then
    MsgBox("Open text file for reading fails.")
    return false
  end

  text=b.Read(b.length)
  err=b.lastErrorCode

  b.Close()

  if err<>0 then
    MsgBox("Read text file fails.")
    return false
  end

  if b.lastErrorCode<>0 then
    MsgBox("Close text file fails.")
    return false
  end

  return true
End Function
```

FileUtils.WriteText:

```
Protected Function WriteText(spec as FolderItem,text as string) As boolean
  dim t as TextOutputStream
  dim err as integer

  t=spec.CreateTextFile()
  if t.lastErrorCode<>0 then
    MsgBox("Create text file fails.")
    return false
  end

  t.Write(text)
  err=t.lastErrorCode

  t.Close()

  if err<>0 then
    MsgBox("Write text file fails.")
    return false
  end
```

```
  if t.lastErrorCode<>0 then
    MsgBox("Close text file fails.")
    return false
  end

  return true
End Function
```

TextUtils (Module)

Constants

String DQUOTE = """
String STYLE_INFO = "<style>.+</style>|<script.+</script>|<(base|div|form|input|link|meta|span) [^>]+>|</(div|form|span)>| (class|data-file-height|data-file-type|data-file-width|id|scope|style|title)="[^"]*"

Properties

m_r as RegEx

TextUtils.SplitURL:

```
Protected Sub SplitURL(url as string,byref path as string,byref label as string,byref extension as stri
              ng,byref  isPage as boolean)
  dim pos_question as integer
  dim pos_cardinal as integer
  dim test as integer
  dim pos_dot as integer

  path=url
  pos_question=path.Instr("?")
  if pos_question>0 then
    path=path.Left(pos_question-1)
  end

  pos_cardinal=path.Instr("#")
  if pos_cardinal>0 then
    path=path.Left(pos_cardinal-1)
    label=path.Mid(pos_cardinal)
  end

  test=path.Instr(".")
  if test=0 then
    return
  end

  do
    pos_dot=test
    test=path.Instr(test+1,".")
  loop until test=0

  extension=path.Mid(pos_dot)
  select case extension
  case ".html"
    extension=".htm"
  case ".jpeg"
    extension=".jpg"
  end
```

```
  extension=extension.Lowercase()

  isPage=(extension=".htm")
End Sub
```

TextUtils.DeleteEmptyStrings:

```
Protected Sub DeleteEmptyStrings(byref lines() as string)
  dim i as integer
  dim s as string

  i=Ubound(lines)
  while i>-1
    s=lines(i)
    if s="" then
      lines.Remove(i)
    end
    i=i-1
  wend
End Sub
```

TextUtils.MakePath:

```
Protected Function MakePath(nr as integer,extension as string) As string
  dim a as integer
  dim b as integer
  dim c as integer
  dim d as integer
  dim a_string as string
  dim b_string as string
  dim c_string as string
  dim d_string as string
  dim path as string

  a=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  b=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  c=Bitwise.BitAnd(nr,255)
  nr=Bitwise.ShiftRight(nr,8)
  d=Bitwise.BitAnd(nr,255)

  a_string=Str(a)+extension
  b_string=Str(b)
  c_string=Str(c)
  d_string=Str(d)
  select case extension
  case ".jpg",".gif",".png"
    d_string="images/"+d_string
  end
```

```
  path=d_string+"/"+c_string+"/"+b_string+"/"+a_string

  return path
End Function
```

TextUtils.StartsWith:

```
Protected Function StartsWith(s as string,t as string) As boolean
  dim s_length as integer
  dim t_length as integer
  dim u as string

  s_length=s.Len()
  t_length=t.Len()
  if s_length<t_length then
    return false
  end
  u=s.Left(t_length)
  if u<>t then
    return false
  end
  return true
End Function
```

TextUtils.IsH2:

```
Protected Function IsH2(line as string,byref name as string) As boolean
  dim pos0 as integer
  dim pos1 as integer

  if not StartsWith(line,"<h2") then
    return false
  end

  pos0=line.InStr(">")
  if pos0=0 then
    return false
  end

  pos1=line.InStr("</h2>")
  if pos1=0 then
    return false
  end

  name=line.Mid(pos0+1,pos1-pos0-1)
  name=name.Trim()

  return true
End Function
```

TextUtils.DQ:

```
Function DQ(s as string) As string
  dim t as string

  t=DQUOTE+s+DQUOTE
  return t
End Function
```

TextUtils.Init:

```
Protected Function Init() As boolean
  m_r=new RegEx
  if m_r=nil then
    MsgBox("Out of memory.")
    return false
  end
  m_r.options.dotMatchAll=true
  m_r.options.greedy=false
  m_r.options.replaceAllMatches=true
  m_r.options.treatTargetAsOneLine=true

  // note: Kiwix files are UNIX files.
  m_r.options.lineEndType=4

  m_r.searchPattern=STYLE_INFO

  return true
End Function
```

TextUtils.DeleteStyle:

```
Protected Function DeleteStyle(s as string,byref t as string) As boolean
  t=m_r.Replace(s)
  return true
catch err as RegExException
  MsgBox("Search fails.")
  return false
End Function
```

Delayed (Class)

Delayed.Delayed_Check:
Sub Delayed_Check()

End Sub

Delayed.Delayed_Set_isUpdated:
Sub Delayed_Set_isUpdated()

End Sub

Delayed.Delayed_Set_hasChanged:
Sub Delayed_Set_hasChanged()

End Sub

Delayed.Delayed_Init:
Sub Delayed_Init()

End Sub

Dumpable (Class)

Dumpable.Dump_WriteLn:

```
Function Dump_WriteLn(r as Report) As boolean

End Function
```

DelayBar (Class)

Properties

```
m_hasChanged as boolean


m_max as integer


m_value as integer
```

DelayBar.Delayed_Check:

```
Sub Delayed_Check()
  dim pos as integer

  if not m_hasChanged then
    return
  end

  if m_max=0 then
    return
  end

  if m_value>m_max then
    // note: Programming mistake.
    return
  end

  // note: integer division is faster.
  pos=m_value*maximum\m_max

  if pos<>value then
    value=pos
  end

  Delayed_Set_isUpdated()

  App.DoEvents()
End Sub
```

DelayBar.Delayed_Set_hasChanged:

```
Private Sub Delayed_Set_hasChanged()
  m_hasChanged=true
End Sub
```

DelayBar.Delayed_Set_isUpdated:

```
Private Sub Delayed_Set_isUpdated()
  m_hasChanged=false
End Sub
```

DelayBar.Set_max:

```
Sub Set_max(max as integer)
  m_max=max
  Delayed_Set_hasChanged()
End Sub
```

DelayBar.Set_value:

```
Sub Set_value(value as integer)
  m_value=value
  Delayed_Set_hasChanged()
End Sub
```

DelayBar.Delayed_Init:

```
Sub Delayed_Init()
  Set_max(1)
  Set_value(0)
  Delayed_Check()
End Sub
```

DelayBar.Inc_value:

```
Sub Inc_value()
  m_value=m_value+1
  Delayed_Set_hasChanged()
End Sub
```

DelayBar.Resized:

```
Sub Resized()
  maximum=width
End Sub
```

DelayBar.Open:

```
Sub Open()
  maximum=width
End Sub
```

DelayNumber (Class)

Properties
m_value as integer

DelayNumber.Inc_value:
```
Sub Inc_value()
  dim value as integer

  value=m_value+1
  Set_value(value)
End Sub
```

DelayNumber.Delayed_Init:
```
Sub Delayed_Init()
  Set_value(0)
  Delayed_Check()
End Sub
```

DelayNumber.Set_value:
```
Sub Set_value(value as integer)
  dim s as string

  m_value=value
  s=Str(m_value)
  Set_text(s)
End Sub
```

DelayNumber.Get_value:
```
Function Get_value() As integer
  return m_value
End Function
```

DelayText (Class)

Properties

```
m_hasChanged as boolean


m_text as string
```

DelayText.Delayed_Check:

```
Sub Delayed_Check()
  if not m_hasChanged then
    return
  end

  if m_text<>text then
    text=m_text
  end

  Delayed_Set_isUpdated()

  App.DoEvents()
End Sub
```

DelayText.Delayed_Set_hasChanged:

```
Private Sub Delayed_Set_hasChanged()
  m_hasChanged=true
End Sub
```

DelayText.Delayed_Set_isUpdated:

```
Private Sub Delayed_Set_isUpdated()
  m_hasChanged=false
End Sub
```

DelayText.Set_text:

```
Sub Set_text(s as string)
  m_text=s
  Delayed_Set_hasChanged()
End Sub
```

DelayText.Delayed_Init:

```
Sub Delayed_Init()
  Set_text("")
  Delayed_Check()
End Sub
```

DelayText.Get_text:

```
Function Get_text() As string
  return m_text
End Function
```

KiwixSocket (Class)

Properties

```
m_isPage as boolean


m_spec as FolderItem


m_stop as boolean


m_absoluteURL as string
```

KiwixSocket.StartDownload:

```
Sub StartDownload(absoluteURL as string,spec as FolderItem,isPage as boolean)
  m_absoluteURL=absoluteURL
  m_spec=spec
  m_isPage=isPage

  m_stop=false

  ZimWindow.Notify_Sock_DownloadStarted(absoluteURL)
  Get(absoluteURL,spec)
End Sub
```

KiwixSocket.Stop:

```
Sub Stop()
  m_stop=true
  if isConnected then
    Disconnect()
  end
  Close()
End Sub
```

KiwixSocket.ReceiveProgress:

```
Sub ReceiveProgress(bytesReceived as integer, totalBytes as integer, newData as string)
  if m_stop then
    return
  end
  ZimWindow.Sock_ReceiveProgress(bytesReceived,totalBytes)
End Sub
```

KiwixSocket.Error:

```
Sub Error(code as integer)
  dim err as string

  if m_stop then
    return
  end
  Stop()
  err="Download "+DQ(m_absoluteURL)+" fails (error "+Str(code)+")."
  ZimWindow.Sock_Error(m_isPage,err)
End Sub
```

KiwixSocket.DownloadComplete:

```
Sub DownloadComplete(url as string, httpStatus as integer, headers as internetHeaders, file as fold
                     erItem)
  if m_stop then
    return
  end
  Stop()
  ZimWindow.Sock_DownloadComplete(m_isPage,m_spec)
End Sub
```

SelectAllField (Class)

SelectAllField.EnableMenuItems:

```
Sub EnableMenuItems()
  editSelectAll.enabled=true
End Sub
```

SelectAllField.EditSelectAll:

```
Function Action() As Boolean
  dim length as integer

  selStart=0
  length=text.Len()
  selLength=length
  return true
End Function
```

AVLNode (Class)

Properties

m_left as AVLNode

m_right as AVLNode

m_balance as integer

m_key as string

m_value as integer

m_height as integer

AVLNode.Verify:

```
Function Verify() As boolean
  dim height as integer
  dim balance as integer

  CalcHeightAndBalance(height,balance)
  if height<>m_height then
    MsgBox("Height is different.")
    return false
  end

  if balance<>m_balance then
    MsgBox("Balance is different.")
    return false
  end

  if m_left<>nil then
    if m_key<=m_left.m_key then
      MsgBox("m_key<=m_left.m_key.")
      return false
    end
```

```
    if not m_left.Verify() then
      MsgBox("Verify left fails.")
      return false
    end
  end

  if m_right<>nil then
    if m_key>=m_right.m_key then
      MsgBox("m_key>=m_right.m_key.")
      return false
    end
    if not m_right.Verify() then
      MsgBox("Verify right fails.")
      return false
    end
  end

  return true
End Function
```

AVLNode.CalcHeightAndBalance:

```
Private Sub CalcHeightAndBalance(byref height as integer,byref balance as integer)
  dim leftHeight as integer
  dim rightHeight as integer

  if m_left<>nil then
    leftHeight=m_left.m_height
  end

  if m_right<>nil then
    rightHeight=m_right.m_height
  end

  if leftHeight>rightHeight then
    height=1+leftHeight
  else
    height=1+rightHeight
  end

  balance=rightHeight-leftHeight
End Sub
```

AVLNode.UpdateHeightAndBalance:

```
Sub UpdateHeightAndBalance()
  CalcHeightAndBalance(m_height,m_balance)
End Sub
```

AVLNode.Find:

```
Function Find(key as string,byref value as integer) As boolean
  if key<m_key then
    if m_left<>nil then
      if m_left.Find(key,value) then
        return true
      end
    end
    return false
  end

  if key>m_key then
    if m_right<>nil then
      if m_right.Find(key,value) then
        return true
      end
    end
    return false
  end

  value=m_value

  return true
End Function
```

AVLNode.Constructor:

```
Sub Constructor(key as string,value as integer)
  m_height=1
  m_key=key
  m_value=value
End Sub
```

AVLNode.Get_key:

```
Function Get_key() As string
  return m_key
End Function
```

AVLNode.Get_left:

```
Function Get_left() As AVLNode
  return m_left
End Function
```

AVLNode.Set_left:

```
Sub Set_left(left as AVLNode)
  m_left=left
End Sub
```

AVLNode.Get_balance:

```
Function Get_balance() As integer
  return m_balance
End Function
```

AVLNode.Get_right:

```
Function Get_right() As AVLNode
  return m_right
End Function
```

AVLNode.Set_right:

```
Sub Set_right(right as AVLNode)
  m_right=right
End Sub
```

AVLNode.Dump_WriteLn:

```
Function Dump_WriteLn(r as Report) As boolean
  dim path as string
  dim label as string
  dim extension as string
  dim isPage as boolean
  dim translatedPath as string
  dim s as string

  if m_left<>nil then
    if not m_left.Dump_WriteLn(r) then
      MsgBox("Dump left fails.")
      return true
    end
  end

  s="old="+DQ(m_key)
  if m_value>=0 then
    TextUtils.SplitURL(m_key,path,label,extension,isPage)
    translatedPath=TextUtils.MakePath(m_value ,extension)
    s=s+" new="+DQ(translatedPath)+"."
  end
  if not r.WriteLn(s) then
    MsgBox("Write to report fails.")
    return false
  end

  if m_right<>nil then
    if not m_right.Dump_WriteLn(r) then
      MsgBox("Dump right fails.")
      return true
    end
  end
```

```
  return true
End Function
```

AVLTree (Class)

Properties
m_top as AVLNode

AVLTree.Verify:
```
Function Verify() As boolean
  if m_top=nil then
    return true
  end

  if not m_top.Verify() then
    MsgBox("Verify top fails.")
    return false
  end

  return true
End Function
```

AVLTree.Find:
```
Function Find(key as string,byref value as integer) As boolean
  dim t as integer

  if m_top=nil then
    return false
  end

  if not m_top.Find(key,value) then
    return false
  end

  return true
End Function
```

AVLTree.Insert:
```
Function Insert(key as string,value as integer) As boolean
  if m_top=nil then
    m_top=new AVLNode(key,value)
    if m_top=nil then
      MsgBox("Out of memory.")
      return false
    end
    return true
  end
```

```
  if not AVLUtils.Insert(m_top,key,value) then
    MsgBox("Insert fails.")
    return false
  end

  return true
End Function
```

AVLTree.Dump_WriteLn:

```
Function Dump_WriteLn(r as Report) As boolean
  if m_top=nil then
    return true
  end

  if not m_top.Dump_WriteLn(r) then
    MsgBox("Dump top fails.")
    return false
  end

  return true
End Function
```

AVLUtils (Module)

AVLUtils.Insert:

```
Protected Function Insert(byref top as AVLNode,key as string,value as integer) As boolean
  dim top_key as string
  dim balance as integer
  dim left as AVLNode
  dim right as AVLNode
  dim s as string

  top_key=top.Get_key()

  if key<top_key then
    left=top.Get_left()
    if left<>nil then
      if not Insert(left,key,value) then
        MsgBox("Insert fails.")
        return false
      end
      top.Set_left(left)
    else
      left=new AVLNode(key,value)
      if left=nil then
        MsgBox("Out of memory.")
        return false
      end
      top.Set_left(left)
    end
    top.UpdateHeightAndBalance()
    balance=top.Get_balance()
    if balance<-1 then
      BalanceLeft(top)
    end
    return true
  end

  if key>top_key then
    right=top.Get_right()
    if right<>nil then
      if not Insert(right,key,value) then
        MsgBox("Insert fails.")
        return false
      end
      top.Set_right(right)
    else
      right=new AVLNode(key,value)
      if right=nil then
        MsgBox("Out of memory.")
```

```
        return false
      end
      top.Set_right(right)
    end
    top.UpdateHeightAndBalance()
    balance=top.Get_balance()
    if balance>1 then
      BalanceRight(top)
    end
    return true
  end

  s="Key "+DQ(key)+" is double."
  MsgBox(s)

  return false
End Function
```

AVLUtils.BalanceLeft:

```
Private Sub BalanceLeft(byref k as AVLNode)
  dim left as AVLNode
  dim left_balance as integer

  left=k.Get_left()
  left_balance=left.Get_balance()
  if left_balance<0 then
    RotateToTheRight(k)
  elseif left_balance>0 then
    RotateToTheLeft(left)
    k.Set_left(left)
    RotateToTheRight(k)
  end
End Sub
```

AVLUtils.BalanceRight:

```
Private Sub BalanceRight(byref k as AVLNode)
  dim right as AVLNode
  dim right_balance as integer

  right=k.Get_right()
  right_balance=right.Get_balance()
  if right_balance>0 then
    RotateToTheLeft(k)
  elseif right_balance<0 then
    RotateToTheRight(right)
    k.Set_right(right)
    RotateToTheLeft(k)
  end
End Sub
```

AVLUtils.RotateToTheLeft:

```
Private Sub RotateToTheLeft(byref k as AVLNode)
  dim r as AVLNode
  dim left as AVLNode
  dim right as AVLNode

  r=k
  k=k.Get_right()
  left=k.Get_left()
  r.Set_right(left)
  k.Set_left(r)

  left=k.Get_left()
  left.UpdateHeightAndBalance()

  right=k.Get_right()
  if right<>nil then
    right.UpdateHeightAndBalance()
  end

  k.UpdateHeightAndBalance()
End Sub
```

AVLUtils.RotateToTheRight:

```
Private Sub RotateToTheRight(byref k as AVLNode)
  dim r as AVLNode
  dim left as AVLNode
  dim right as AVLNode

  r=k
  k=k.Get_left()
  right=k.Get_right()
  r.Set_left(right)
  k.Set_right(r)

  right=k.Get_right()
  right.UpdateHeightAndBalance()

  left=k.Get_left()
  if left<>nil then
    left.UpdateHeightAndBalance()
  end

  k.UpdateHeightAndBalance()
End Sub
```

AVLUtils.Parameters:

Parameters
You can't pass an expression as a "byref" parameter.

x.m_left is an expression.

Brain (Class)

Properties

m_links as Links

m_address as string

m_sock as KiwixSocket

m_conversion as Conversion

m_pages as Pages

m_ls as LinkSearcher

m_fileRequested as boolean

m_deleteStyle as boolean

m_deleteTemp as boolean

m_name as string

m_port as integer

m_address as string

Brain.Init:

```
Function Init() As boolean
  m_links=new Links
  if m_links=nil then
    MsgBox("Out of memory.")
    return false
  end
  if not m_links.Init() then
    MsgBox("Init links fails.")
    return false
  end

  m_ls=new LinkSearcher
  if m_ls=nil then
    MsgBox("Out of memory.")
    return false
  end
  if not m_ls.Init() then
    MsgBox("Init link searcher fails.")
    return false
  end

  m_pages=new Pages
  if m_pages=nil then
    MsgBox("Out of memory.")
    return false
  end

  m_sock=new KiwixSocket
  if m_sock=nil then
    MsgBox("Out of memory.")
    return false
  end

  return true
End Function
```

Brain.Work:

```
Function Work(byref isDone as boolean) As boolean
  dim ok as boolean

  if not m_pages.Get_isEmpty() then
    isDone=false
    if not StartNewConversion() then
      MsgBox("Start new conversion fails.")
      return false
```

```
    end
    return true
  end

  if not m_links.Get_isEmpty() then
    isDone=false
    if not StartNewDownload() then
      MsgBox("Start new download fails.")
      return false
    end
    return true
  end

  isDone=true

  return true
End Function
```

Brain.Constructor:

```
Sub Constructor(address as string,port as integer,name as string,deleteStyle as boolean,deleteTem
            p as boolean)
  m_address=address
  m_deleteStyle=deleteStyle
  m_deleteTemp=deleteTemp
  m_name=name
  m_port=port
End Sub
```

Brain.DownloadIsDone:

```
Sub DownloadIsDone(isPage as boolean,spec as FolderItem)
  m_fileRequested=false
  if isPage then
    m_pages.Append(spec)
  end
End Sub
```

Brain.MakeAbsoluteURL:

```
Private Function MakeAbsoluteURL(relativeURL as string) As string
  dim absoluteURL as string

  if TextUtils.StartsWith(relativeURL,"../") then
    absoluteURL="http://"+m_address+":"+Str(m_port)+"/"+m_name+"/"+relativeURL.Mid(4)
  else
    absoluteURL="http://"+m_address+":"+Str(m_port)+"/"+m_name+"/A/"+relativeURL
  end
  return absoluteURL
End Function
```

Brain.StartNewDownload:

```
Private Function StartNewDownload() As boolean
  dim relativeURL as string
  dim nr as integer
  dim path as string
  dim label as string
  dim extension as string
  dim isPage as boolean
  dim spec as FolderItem
  dim absoluteURL as string

  m_links.Pop(relativeURL,nr)

  TextUtils.SplitURL(relativeURL,path,label,extension,isPage)
  if isPage then
    extension=".tmp"
  end
  if not FileUtils.MakeSpec(nr,extension,spec) then
    MsgBox("Make spec fails.")
    return false
  end

  absoluteURL=MakeAbsoluteURL(relativeURL)

  m_fileRequested=true
  m_sock.StartDownload(absoluteURL,spec,isPage)

  return true
End Function
```

Brain.ConversionIsDone:

```
Sub ConversionIsDone()
  m_conversion=nil
End Sub
```

Brain.StartNewConversion:

```
Private Function StartNewConversion() As boolean
  dim spec as FolderItem
  spec=m_pages.Pop()

  m_conversion=new Conversion(spec,m_links,m_ls,m_deleteStyle,m_deleteTemp)
  if m_conversion=nil then
    MsgBox("Out of memory.")
    return false
  end
  m_conversion.Run()

  return true
End Function
```

Brain.DownloadError:

```
Sub DownloadError()
  m_fileRequested=false
End Sub
```

Brain.IsBusy:

```
Function IsBusy() As boolean
  if m_fileRequested then
    return true
  end

  if m_conversion<>nil then
    return true
  end

  return false
End Function
```

Brain.Dump_WriteLn:

```
Function Dump_WriteLn(r as Report) As boolean
  if not m_links.Dump_WriteLn(r) then
    MsgBox("Dump fails.")
    return false
  end
  return true
End Function
```

Conversion (Class)

Constants
String TABLE_NAVBOX = "<table class="navbox" "
String CLASS_EXTERNAL = " class="external""

Properties
m_input as FolderItem

m_linksRef as Links

m_lsRef as LinkSearcher

m_deleteStyle as boolean

m_deleteTemp as boolean

Conversion.Constructor:
```
Sub Constructor(input as FolderItem,links as Links,ls as LinkSearcher,deleteStyle as boolean,delete
        Temp as boolean)
  m_deleteStyle=deleteStyle
  m_deleteTemp=deleteTemp
  m_input=input
  m_linksRef=links
  m_lsRef=ls
End Sub
```

Conversion.TranslateLine:
```
Private Function TranslateLine(line as string,byref translatedLine as string,byref deleteRest as bool
        ean) As boolean
  dim name as string
  dim pos as integer

  // note: The base tag contains a link which can contain a double quote.
  if TextUtils.StartsWith(line,"<base ") then
    translatedLine=""
    return true
  end
```

```
  if TextUtils.IsH2(line,name) then
    select case name
    case "References","External links","Notes","See also","Further reading"
      deleteRest=true
      return true
    end
  end

  if TextUtils.StartsWith(line,TABLE_NAVBOX) then
    deleteRest=true
    return true
  end

  // note: An average of 1 in 3 lines contains an average of 3 links.
  // This extra filter will accelerate the conversion.
  pos=line.InStr("href=")
  if pos=0 then
    pos=line.InStr("src=")
    if pos=0 then
      translatedLine=line
      return true
    end
  end

  if not TranslateLinks(line,translatedLine) then
    MsgBox("Translate links fails.")
    return false
  end

  return true
End Function
```

Conversion.TranslateLinks:

```
Private Function TranslateLinks(line as string,byref translatedLine as string) As boolean
  dim length as integer

  dim pos as integer
  dim found as boolean
  dim sub0_pos as integer
  dim sub0_length as integer
  dim sub1 as string
  dim sub2 as string
  dim isExternal as boolean

  dim skipped as string
  dim parts() as string

  dim translatedLink as string
```

```
  dim rest as string

  length=line.Len()
  pos=1

  if not m_lsRef.SearchLink(line,pos,found,sub0_pos,sub0_length,sub1,sub2,isExternal) then
    MsgBox("Search link fails.")
    return false
  end
  while found
    if sub0_pos>pos then
      skipped=line.Mid(pos,sub0_pos-pos)
      parts.Append(skipped)
    end

    if not TranslateLink(sub1,sub2,isExternal,translatedLink) then
      MsgBox("Translate link fails.")
      return false
    end

    parts.Append(translatedLink)

    pos=sub0_pos+sub0_length

    if pos+11>length then
      found=false
    elseif not m_lsRef.SearchLink(line,pos,found,sub0_pos,sub0_length,sub1,sub2,isExternal) then
      MsgBox("Search link fails.")
      return false
    end
  wend

  if pos<=length then
    rest=line.Mid(pos)
    parts.Append(rest)
  end

  translatedLine=Join(parts,"")

  return true
End Function
```

Conversion.ConvertText:

```
Private Function ConvertText(text as string,byref convertedText as string) As boolean
  dim lines() as string
  dim cLines as integer
  dim line as string
  dim translatedLine as string
  dim deleteRest as boolean
```

```
  dim translatedLines() as string

  // note: Kiwix files are UNIX files.
  lines=text.Split(EndOfLine.UNIX)

  cLines=UBound(lines)+1
  ZimWindow.Notify_Conversion_TranslationStarted(cLines)

  for each line in lines
    if line="" then
      translatedLines.Append("")
    else
      if not TranslateLine(line,translatedLine,deleteRest) then
        MsgBox("Translate line fails.")
        return false
      end
      if deleteRest then
        // note: All what follows will be garbage that I don't want.
        translatedLines.Append("</body></html>")
        convertedText=Join(translatedLines,EndOfLine.UNIX)
        return true
      end
      translatedLines.Append(translatedLine)
      translatedLine=""
    end
    ZimWindow.Notify_Conversion_LineTranslated()
  next

  convertedText=Join(translatedLines,EndOfLine.UNIX)

  return true
End Function
```

Conversion.Convert:

```
Private Function Convert() As boolean
  dim text as string
  dim convertedText as string
  dim name as string
  dim output as FolderItem
  dim err as integer

  ZimWindow.Notify_Conversion_ConversionStarted(m_input.absolutePath)

  if not FileUtils.ReadText(m_input,text) then
    MsgBox("Read text file fails.")
    return false
  end

  if m_deleteStyle then
```

```
    if not TextUtils.DeleteStyle(text,text) then
      MsgBox("Optimize for Palm OS fails.")
      return false
    end
  end

  if not ConvertText(text,convertedText) then
    MsgBox("Convert text fails.")
    return false
  end

  name=m_input.name.Replace(".tmp",".htm")
  output=m_input.parent.TrueChild(name)
  if output=nil then
    MsgBox("Out of memory.")
    return false
  end
  if not FileUtils.WriteText(output,convertedText) then
    MsgBox("Write text file fails.")
    return false
  end

  if m_deleteTemp then
    m_input.Delete()
    err=m_input.lastErrorCode
    if err<>0 then
      MsgBox("Delete temp fails.")
      return false
    end
  end

  return true
End Function
```

Conversion.TranslateLink:

```
Private Function TranslateLink(attribute as string,relativeURL as string,isExternal as boolean,byref t
                ranslatedLink as string) As boolean
  dim path as string
  dim label as string
  dim extension as string
  dim isPage as boolean
  dim nr as integer
  dim translatedPath as string

  if isExternal then
    translatedLink=CLASS_EXTERNAL
    return true
  end
```

```
  if TextUtils.StartsWith(relativeURL,"/") then
    relativeURL=".."+relativeURL
  end

  TextUtils.SplitURL(relativeURL,path,label,extension,isPage)

  if path="" then
    translatedLink=" "+attribute+"="+DQ(label)
    return true
  end

  if extension=".htm" then
    if m_linksRef.Recall(path,extension,nr) then
      translatedPath=TextUtils.MakePath(nr,extension)
      translatedLink=" "+attribute+"="+DQ("../../../"+translatedPath+label)
    else
      translatedLink=CLASS_EXTERNAL
      if not m_linksRef.StoreExternalIfNew(relativeURL) then
        MsgBox("Store external fails.")
        return false
      end
    end
    return true
  end

  if not m_linksRef.Recall(path,extension,nr) then
    if not m_linksRef.StoreAndAppend(path,extension,nr) then
      MsgBox("Store and append fails.")
      return false
    end
    ZimWindow.Notify_Conversion_OtherAppended()
  end
  translatedPath=TextUtils.MakePath(nr,extension)
  translatedLink=" "+attribute+"="+DQ("../../../"+translatedPath)

  return true
End Function
```

Conversion.Run:

```
Sub Run()
  if not Convert() then
    MsgBox("Convert file fails.")
    ZimWindow.Conversion_Fail()
    return
  end

  ZimWindow.Conversion_Complete()
End Sub
```

Links (Class)

Properties

m_pages() as AVLTree

m_relativeURLs() as string

m_nr as integer

m_nrs() as integer

m_jsCSS as AVLTree

m_others as AVLTree

m_externalPages as AVLTree

Links.Init:

```
Function Init() As boolean
  dim i as integer
  dim n as AVLTree
  dim crelativeURLs as integer

  for i=0 to 675
    n=new AVLTree
    if n=nil then
      MsgBox("Out of memory.")
      return false
    end
    m_pages.Append(n)
  next

  m_jsCSS=new AVLTree
  if m_jsCSS=nil then
```

```
    MsgBox("Out of memory.")
    return false
  end

  m_others=new AVLTree
  if m_others=nil then
    MsgBox("Out of memory.")
    return false
  end

  m_externalPages=new AVLTree
  if m_externalPages=nil then
    MsgBox("Out of memory.")
    return false
  end

  if not ReadLinks() then
    MsgBox("Read links fails.")
    return false
  end

  crelativeURLs=Ubound(m_relativeURLs)+1
  ZimWindow.Notify_Links_Read(crelativeURLs)

  return true
End Function
```

Links.Store:

```
Private Function Store(relativeURL as string,extension as string) As boolean
  dim t as AVLTree

  t=FindTree(relativeURL,extension)
  if not t.Insert(relativeURL,m_nr) then
    MsgBox("Insert fails.")
    return false
  end
  m_nr=m_nr+1
  return true
End Function
```

Links.ReadLinks:

```
Private Function ReadLinks() As boolean
  dim spec as FolderItem
  dim text as string
  dim relativeURLs() as string
  dim relativeURL as string
  dim path as string
  dim label as string
  dim extension as string
```

```
dim isPage as boolean
dim nr as integer
dim left as integer
dim right as integer
dim s as string
dim r as integer

spec=new FolderItem("")
if spec=nil then
  MsgBox("Out of memory.")
  return false
end

spec=spec.TrueChild("PageURLs.txt")
if spec=nil then
  MsgBox("Out of memory.")
  return false
end
if not spec.exists then
  MsgBox("File 'PageURLs.txt' not found.")
  return false
end

if not FileUtils.ReadText(spec,text) then
  MsgBox("Read text file fails.")
  return false
end

// note: Be sure that the EOL matches your text file.
relativeURLs=text.Split(EndOfLine)
TextUtils.DeleteEmptyStrings(relativeURLs)
relativeURLs.Sort()

for each relativeURL in relativeURLs
  TextUtils.SplitURL(relativeURL,path,label,extension,isPage)
  if not StoreAndAppend(relativeURL,extension,nr) then
    MsgBox("Store fails.")
    return false
  end
next

// note: Push in reverse order => pop in order.
left=0
right=Ubound(m_relativeURLs)
while left<right
  s=m_relativeURLs(left)
  m_relativeURLs(left)=m_relativeURLs(right)
  m_relativeURLs(right)=s
```

```
    r=m_nrs(left)
    m_nrs(left)=m_nrs(right)
    m_nrs(right)=r

    left=left+1
    right=right-1
  wend

  return true
End Function
```

Links.Recall:

```
Function Recall(relativeURL as string,extension as string,byref nr as integer) As boolean
  dim t as AVLTree

  t=FindTree(relativeURL,extension)
  if not t.Find(relativeURL,nr) then
    return false
  end
  return true
End Function
```

Links.Pop:

```
Sub Pop(byref relativeURL as string,byref nr as integer)
  relativeURL=m_relativeURLs.Pop()
  nr=m_nrs.Pop()
End Sub
```

Links.Get_isEmpty:

```
Function Get_isEmpty() As boolean
  dim hrelativeURLs as integer

  hrelativeURLs=Ubound(m_relativeURLs)
  if hrelativeURLs>=0 then
    return false
  end
  return true
End Function
```

Links.StoreAndAppend:

```
Function StoreAndAppend(relativeURL as string,extension as string,byref nr as integer) As boolean
  nr=m_nr
  if not Store(relativeURL,extension) then
    MsgBox("Store fails.")
    return false
  end
  m_relativeURLs.Append(relativeURL)
  m_nrs.Append(nr)
  return true
```

```
End Function
```

Links.CalcPageTreeID:

```
Private Function CalcPageTreeID(relativeURL as string) As integer
  dim c as string
  dim a as integer
  dim id as integer

  c=relativeURL.Mid(1,1)
  a=Asc(c)
  a=a mod 26
  id=26*a

  c=relativeURL.Mid(2,1)
  a=Asc(c)
  a=a mod 26
  id=id+a

  return id
End Function
```

Links.Dump_WriteLn:

```
Function Dump_WriteLn(r as Report) As boolean
  dim t as AVLTree
  dim external as string

  if not r.WriteLn("========== HTML pages ==========") then
    MsgBox("Write to report fails.")
    return false
  end
  for each t in m_pages
    if not t.Dump_WriteLn(r) then
      MsgBox("Dump fails.")
      return false
    end
  next

  if not r.WriteLn("========== JavaScript and CSS files ==========") then
    MsgBox("Write to report fails.")
    return false
  end
  if not m_jsCSS.Dump_WriteLn(r) then
    MsgBox("Dump fails.")
    return false
  end

  if not r.WriteLn("========== Other files ==========") then
    MsgBox("Write to report fails.")
    return false
```

```
  end
  if not m_others.Dump_WriteLn(r) then
    MsgBox("Dump fails.")
    return false
  end

  if not r.WriteLn("========== External HTML pages ==========") then
    MsgBox("Write to report fails.")
    return false
  end
  if not m_externalPages.Dump_WriteLn(r) then
    MsgBox("Dump fails.")
    return false
  end

  return true
End Function
```

Links.StoreExternalIfNew:

```
Function StoreExternalIfNew(relativeURL as string) As boolean
  dim nr as integer

  if m_externalPages.Find(relativeURL,nr) then
    return true
  end
  if not m_externalPages.Insert(relativeURL,-1) then
    MsgBox("Insert fails.")
    return false
  end
  return true
End Function
```

Links.FindTree:

```
Private Function FindTree(relativeURL as string,extension as string) As AVLTree
  dim id as integer
  dim t as AVLTree

  select case extension
  case ".htm"
    id=CalcPageTreeID(relativeURL)
    t=m_pages(id)
  case ".js",".css"
    t=m_jsCSS
  else
    t=m_others
  end
  return t
End Function
```

LinkSearcher (Class)

Constants
String HREF_SRC = " (href|src)="([^"]+)""

Properties
m_r as RegEx

m_s as RegEx

LinkSearcher.Init:
```
Function Init() As boolean
  m_r=new RegEx
  if m_r=nil then
    MsgBox("Out of memory.")
    return false
  end
  m_r.searchPattern=HREF_SRC

  m_s=new RegEx
  if m_s=nil then
    MsgBox("Out of memory.")
    return false
  end
  m_s.searchPattern="https://|http://|geo:|ftp://|upload.wikimedia.org"

  return true
End Function
```

LinkSearcher.SearchLink:
```
Function SearchLink(line as string,pos as integer,byref found as boolean,byref sub0_pos as integer,b
            yref sub0_length as integer,byref sub1 as string,byref sub2 as string,byref isExternal as
            boolean) As boolean
  dim m as RegExMatch
  dim sub0 as string
  dim pos_slash as integer

  // note: Although HTML is not a regular language
  // a regular expression is a good way to search links
  // because the HTML was written by Kiwix.
  m=m_r.Search(line,pos)
  if m=nil then
    found=false
    return true
```

```
  end

  found=true

  // note: String positions are 1-based,
  // but SubExpressionStartB is 0-based
  // so I have to add 1.
  sub0_pos=m.SubExpressionStartB(0)+1

  sub0=m.SubExpressionString(0)
  sub0_length=sub0.Len()

  sub1=m.SubExpressionString(1)
  sub2=m.SubExpressionString(2)

  // note: External links contain a slash.
  // Most links are not external.
  // This extra filter will accelerate the conversion.
  pos_slash=line.InStr("/")
  if pos_slash=0 then
    return true
  end

  m=m_s.Search(sub2)
  isExternal=(m<>nil)

  return true

catch err as RegExException
  MsgBox("Search fails.")
  return false
End Function
```

Pages (Class)

Properties
m_specs() as FolderItem

Pages.Get_isEmpty:
```
Function Get_isEmpty() As boolean
  dim hSpecs as integer

  hSpecs=Ubound(m_specs)
  if hSpecs>=0 then
    return false
  end
  return true
End Function
```

Pages.Pop:
```
Function Pop() As FolderItem
  dim spec as FolderItem

  spec=m_specs.Pop()
  return spec
End Function
```

Pages.Append:
```
Sub Append(spec as FolderItem)
  m_specs.Append(spec)
End Sub
```

Report (Class)

Properties
m_t as TextOutputStream

Report.CreateAndOpen:
```
Function CreateAndOpen() As boolean
  dim spec as FolderItem
  dim err as integer

  spec=new FolderItem("")
  if spec=nil then
    MsgBox("Out of memory.")
    return false
  end

  spec=spec.TrueChild("Report.txt")
  if spec=nil then
    MsgBox("Out of memory.")
    return false
  end
  if spec.exists then
    MsgBox("File 'Report.txt' exists already.")
    return false
  end

  m_t=new TextOutputStream
  if m_t=nil then
    MsgBox("Out of memory.")
    return false
  end

  m_t=spec.CreateTextFile()

  err=m_t.lastErrorCode
  if err<>0 then
    MsgBox("Create stream fails.")
    return false
  end

  return true
End Function
```

Report.WriteLn:

```
Function WriteLn(s as string) As boolean
  dim err as integer

  m_t.WriteLine(s)

  err=m_t.lastErrorCode
  if err<>0 then
    MsgBox("Write line of text fails.")
    return false
  end

  return true
End Function
```

Report.Close:

```
Function Close() As boolean
  dim err as integer

  m_t.Close()

  err=m_t.lastErrorCode
  if err<>0 then
    MsgBox("Close stream fails.")
    return false
  end

  return true
End Function
```

ZimWindow (Window)

Properties

m_brain as Brain

m_report as Report

m_cancel as boolean

m_stop as boolean

m_isUpdating as boolean

m_isWorking as boolean

ZimWindow.AbleControls:

```
Private Sub AbleControls(active as boolean)
  SF_Address.enabled=active
  SF_Port.enabled=active
  SF_Name.enabled=active
  CB_DeleteStyle.enabled=active
  CB_DeleteTemp.enabled=active

  PB_Start.enabled=active
  PB_Cancel.enabled=not active

  DN_PageCount.Delayed_Init()
  DB_PagesDownloaded.Delayed_Init()
  DB_PagesConverted.Delayed_Init()

  DN_OtherCount.Delayed_Init()
  DB_OthersDownloaded.Delayed_Init()

  DT_DownloadingNowURL.Delayed_Init()
  DB_DownloadingNowReceived.Delayed_Init()
```

```
  DT_ConvertingNowPath.Delayed_Init()
  DB_ConvertingNowTranslated.Delayed_Init()
End Sub
```

ZimWindow.Start:

```
Private Function Start() As boolean
  dim address as string
  dim port as integer
  dim name as string
  dim deleteStyle as boolean
  dim deleteTemp as boolean

  m_cancel=false
  m_stop=false

  m_report=new Report
  if m_report=nil then
    MsgBox("Out of memory.")
    return false
  end
  if not m_report.CreateAndOpen() then
    m_report=nil
    MsgBox("Create and open report fails.")
    return false
  end

  address=SF_Address.text
  port=Val(SF_Port.text)
  name=SF_Name.text
  deleteStyle=CB_DeleteStyle.value
  deleteTemp=CB_DeleteTemp.value

  m_brain=new Brain(address,port,name,deleteStyle,deleteTemp)
  if m_brain=nil then
    MsgBox("Out of memory.")
    return false
  end
  if not m_brain.Init() then
    MsgBox("Init brain fails.")
    return false
  end

  TM_Work.mode=2
  TM_Update.mode=2

  return true
End Function
```

ZimWindow.Finalize:

```
Private Sub Finalize(s as string)
  TM_Work.mode=0
  TM_Update.mode=0

  if m_report=nil then
    MsgBox(s)
  elseif not m_report.WriteLn(s) then
    MsgBox(s)
  end

  if m_report<>nil then
    if not m_report.Close() then
      MsgBox("Close report fails.")
    end
    m_report=nil
  end

  AbleControls(true)
  Refresh()
End Sub
```

ZimWindow.Sock_DownloadComplete:

```
Sub Sock_DownloadComplete(isPage as boolean,spec as FolderItem)
  m_brain.DownloadIsDone(isPage,spec)
  if isPage then
    Notify_Sock_PageDownloaded()
  else
    Notify_Sock_OtherDownloaded()
  end
End Sub
```

ZimWindow.Sock_Error:

```
Sub Sock_Error(isPage as boolean,err as string)
  ApplyStop()
  m_brain.DownloadError()
  if isPage then
    Notify_Sock_PageDownloaded()
  else
    Notify_Sock_OtherDownloaded()
  end
  MsgBox(err)
End Sub
```

ZimWindow.Sock_ReceiveProgress:

```
Sub Sock_ReceiveProgress(bytesReceived as integer,totalBytes as integer)
  if bytesReceived>totalBytes then
    // note: Some servers return the wrong length or 0.
    return
```

```
  end
  Notify_Sock_Received(bytesReceived,totalBytes)
End Sub
```

ZimWindow.Conversion_Complete:

```
Sub Conversion_Complete()
  m_brain.ConversionIsDone()
  Notify_Conversion_TranslationComplete()
End Sub
```

ZimWindow.Conversion_Fail:

```
Sub Conversion_Fail()
  Conversion_Complete()
  MsgBox("Conversion fails.")
  ApplyStop()
End Sub
```

ZimWindow.Notify_Conversion_LineTranslated:

```
Sub Notify_Conversion_LineTranslated()
  DB_ConvertingNowTranslated.Inc_value()
End Sub
```

ZimWindow.Notify_Conversion_TranslationStarted:

```
Sub Notify_Conversion_TranslationStarted(cLines as integer)
  DB_ConvertingNowTranslated.Set_max(cLines)
  DB_ConvertingNowTranslated.Set_value(0)
End Sub
```

ZimWindow.Notify_Conversion_OtherAppended:

```
Sub Notify_Conversion_OtherAppended()
  dim cOther as integer

  DN_OtherCount.Inc_value()
  cOther=DN_OtherCount.Get_value()
  DB_OthersDownloaded.Set_max(cOther)
End Sub
```

ZimWindow.Notify_Links_Read:

```
Sub Notify_Links_Read(cLinks as integer)
  DN_PageCount.text=Str(cLinks)
  DB_PagesDownloaded.Set_max(cLinks)
  DB_PagesDownloaded.Set_value(0)
  DB_PagesConverted.Set_max(cLinks)
  DB_PagesConverted.Set_value(0)
End Sub
```

ZimWindow.Notify_Conversion_TranslationComplete:

```
Sub Notify_Conversion_TranslationComplete()
  DB_PagesConverted.Inc_value()
```

```
End Sub
```

ZimWindow.Notify_Sock_PageDownloaded:

```
Sub Notify_Sock_PageDownloaded()
  DB_PagesDownloaded.Inc_value()
End Sub
```

ZimWindow.Notify_Sock_OtherDownloaded:

```
Sub Notify_Sock_OtherDownloaded()
  DB_OthersDownloaded.Inc_value()
End Sub
```

ZimWindow.Notify_Sock_Received:

```
Sub Notify_Sock_Received(bytesReceived as integer,totalBytes as integer)
  DB_DownloadingNowReceived.Set_max(totalBytes)
  DB_DownloadingNowReceived.Set_value(bytesReceived)
End Sub
```

ZimWindow.Notify_Sock_DownloadStarted:

```
Sub Notify_Sock_DownloadStarted(absoluteURL as string)
  DT_DownloadingNowURL.Set_text(absoluteURL)
End Sub
```

ZimWindow.Notify_Conversion_ConversionStarted:

```
Sub Notify_Conversion_ConversionStarted(path as string)
  DT_ConvertingNowPath.Set_text(path)
End Sub
```

ZimWindow.ApplyStop:

```
Private Sub ApplyStop()
  m_stop=true
End Sub
```

ZimWindow.Action_Update:

```
Private Sub Action_Update()
  DN_PageCount.Delayed_Check()
  DB_PagesDownloaded.Delayed_Check()
  DB_PagesConverted.Delayed_Check()

  DN_OtherCount.Delayed_Check()
  DB_OthersDownloaded.Delayed_Check()

  DT_DownloadingNowURL.Delayed_Check()
  DB_DownloadingNowReceived.Delayed_Check()

  DT_ConvertingNowPath.Delayed_Check()
  DB_ConvertingNowTranslated.Delayed_Check()
End Sub
```

ZimWindow.Action_Work:

```
Private Sub Action_Work()
  dim isDone as boolean

  if m_brain.IsBusy() then
    return
  end

  if m_stop then
    if not m_brain.Dump_WriteLn(m_report) then
      MsgBox("Dump brain fails.")
    end
    m_brain=nil
    if m_cancel then
      Finalize("Canceled.")
    else
      Finalize("Done.")
    end
  else
    if not m_brain.Work(isDone) then
      ApplyStop()
      MsgBox("Work fails.")
      return
    end
    if isDone then
      ApplyStop()
      MsgBox("Finished.")
    end
  end
End Sub
```

ZimWindow.Resized:

```
Sub Resized()
  DB_PagesDownloaded.Resized()
  DB_PagesConverted.Resized()

  DB_OthersDownloaded.Resized()

  DB_DownloadingNowReceived.Resized()

  DB_ConvertingNowTranslated.Resized()
End Sub
```

ZimWindow.EnableMenuItems:

```
Sub EnableMenuItems()
  fileQuit.enabled=PB_Start.enabled
End Sub
```

ZimWindow.CancelClose:

```
Function CancelClose(appQuitting as Boolean) As Boolean
  if PB_Start.enabled then
    return false
  end
  MsgBox("Still busy.")
  return true
End Function
```

ZimWindow.PB_Start.Action:

```
Sub Action()
  AbleControls(false)
  if not Start() then
    Finalize("Start fails.")
  end
End Sub
```

ZimWindow.TM_Work.Action:

```
Sub Action()
  if m_isWorking then
    return
  end

  m_isWorking=true
  Action_Work()
  m_isWorking=false
End Sub
```

ZimWindow.TM_Update.Action:

```
Sub Action()
  if m_isUpdating then
    return
  end

  m_isUpdating=true
  Action_Update()
  m_isUpdating=false
End Sub
```

ZimWindow.DT_DownloadingNowURL.Open:

```
Sub Open()
  me.text=""
End Sub
```

ZimWindow.DT_ConvertingNowPath.Open:

```
Sub Open()
  me.text=""
End Sub
```

ZimWindow.PB_Cancel.Action:

```
Sub Action()
  PB_Cancel.enabled=false
  m_cancel=true
  ApplyStop()
End Sub
```

ZimWindow.DN_OtherCount.Open:

```
Sub Open()
  me.text=""
End Sub
```

ZimWindow.DN_PageCount.Open:

```
Sub Open()
  me.text=""
End Sub
```