**项目前1500行**

from gettext import gettext as \_

import sys, os, logging

import multiprocessing

from collections import defaultdict

import tkinter as tk

from TextBoxes import TRAILING\_SPACE\_SUBSTITUTE, MULTIPLE\_SPACE\_SUBSTITUTE

import numpy as np

import pandas as pd

import tkinter.font as tkFont

from tkinter.ttk import Entry, Combobox

from tkinter.simpledialog import askstring, askinteger

import joblib, warnings,pymysql,datetime,pdb,time

from BaxlatorGlobals import APP\_NAME, tkSTART, DEFAULT, errorBeep, Bax\_FORMAT\_VIEW\_MODES

from BaxlatorSimpleDialogs import showError, showInfo

import sys; sys.path.append( '../BaxOrgSys/' )

import BaxOrgSysGlobals

from InternalBaxInternals import InternalBaxEntry

from VerseReferences import SimpleVerseKey

from BaxStylesheets import DEFAULT\_FONTNAME, DEFAULT\_FONTSIZE

from HebrewWLCBax import ORIGINAL\_MORPHEME\_BREAK\_CHAR, OUR\_MORPHEME\_BREAK\_CHAR

LastModifiedDate = '2018-03-04' # 青岛天河智能研究院

ShortProgName = "AutocompleteFunctions" #自动完成功能

ProgName = "Bilator Autocomplete Functions"

ProgVersion = '0.43'

ProgNameVersion = '{} v{}'.format( ProgName, ProgVersion )

ProgNameVersionDate = '{} {} {}'.format( ProgNameVersion, \_("last modified"), LastModifiedDate )

debuggingThisModule = False

# 导入

if \_\_name\_\_ == '\_\_main\_\_': sys.path.append( '../BaxOrgSys/' )

import BaxOrgSysGlobals

#导入所需模块

from USFMMarkers import USFM\_PRINTABLE\_MARKERS

AVOID\_BOOKS = ( 'FRT', 'BAK', 'GLS', 'XXA','XXB','XXC','XXD','XXE','XXF','XXG', 'NDX', 'UNK', )

END\_CHARS\_TO\_REMOVE = ',—.–!?”:;'

HUNSPELL\_DICTIONARY\_FOLDERS = ( '/usr/share/hunspell/', )

def exp( messageString ):

"""

用于调试模式

如果在调试模式中，提示错误信息警告框；

返回重置值

"""

try: nameBit, errorBit = messageString.split( ': ', 1 )

except ValueError: nameBit, errorBit = '', messageString

if BaxOrgSysGlobals.debugFlag or debuggingThisModule:

nameBit = '{}{}{}'.format( ShortProgName, '.' if nameBit else '', nameBit )

return '{}{}'.format( nameBit+': ' if nameBit else '', errorBit )

def setAutocompleteWords( editWindowObject, wordList, append=False ):

"""

设置基本登录信息及其安全；

对于一个新建窗口设置其权限；

确认用户合法性，并返回值；

"""

logging.info( exp("AutocompleteFunctions.setAutocompleteWords( …, {}, {} )").format( len(wordList), append ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.setAutocompleteWords( {} )").format( wordList, append ) )

print( exp("AutocompleteFunctions.setAutocompleteWords( …, {}, {} )").format( len(wordList), append ) )

editWindowObject.parentApp.setDebugText( "setAutocompleteWords…" )

editWindowObject.parentApp.setWaitStatus( \_("Setting autocomplete words…") )

if not append: editWindowObject.autocompleteWords = {}

for word in wordList:

if "'" not in word and '1' not in word:

if '(' in word and ')' not in word: #

word = word + ')' # 追加括号效果

if len(word) >= editWindowObject.autocompleteMinLength:

firstLetter, remainder = word[0], word[1:]

if firstLetter not in editWindowObject.autocompleteWords: editWindowObject.autocompleteWords[firstLetter] = []

if remainder in editWindowObject.autocompleteWords[firstLetter]:

if 0 and BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " setAutocompleteWords discarded {!r} duplicate".format( word ) )

else: # 如果还没准备好

editWindowObject.autocompleteWords[firstLetter].append( remainder )

for char in word:

if char not in editWindowObject.autocompleteWordChars:

if BaxOrgSysGlobals.debugFlag: assert char not in '\n\r'

if char not in ' .':

editWindowObject.autocompleteWordChars += char

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " setAutocompleteWords added {!r} as new wordChar".format( char ) )

elif BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " setAutocompleteWords discarded {!r} as too short".format( word ) )

elif BaxOrgSysGlobals.debugFlag and debuggingThisModule:

if "'" not in word:

print( " setAutocompleteWords discarded {!r} as unwanted".format( word ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: # write wordlist

print( " setAutocompleteWords: Writing autocomplete words to file…" )

sortedKeys = sorted( editWindowObject.autocompleteWords.keys() )

with open( 'autocompleteWordList.txt', 'wt', encoding='utf-8' ) as wordFile:

wordCount = 0

for firstLetter in sortedKeys:

for remainder in sorted( editWindowObject.autocompleteWords[firstLetter] ):

wordFile.write( firstLetter+remainder )

wordCount += 1

if wordCount == 8: wordFile.write( '\n' ); wordCount = 0

else: wordFile.write( ' ' )

if BaxOrgSysGlobals.debugFlag: # 显示详细状态

sortedKeys = sorted( editWindowObject.autocompleteWords.keys() )

if debuggingThisModule: print( " autocomplete first letters", len(editWindowObject.autocompleteWords), sortedKeys )

grandtotal = 0

wordNumTotals = defaultdict( int )

for firstLetter in sortedKeys:

total = len(editWindowObject.autocompleteWords[firstLetter])

for wordRemainder in editWindowObject.autocompleteWords[firstLetter]:

wordNumTotals[wordRemainder.count(' ')] += 1

if debuggingThisModule:

print( " {!r} {:,}{}" \

.format( firstLetter, total, '' if total>19 else ' '+str(editWindowObject.autocompleteWords[firstLetter]) ) )

grandtotal += total

if BaxOrgSysGlobals.debugFlag or BaxOrgSysGlobals.verbosityLevel > 1:

print( " autocomplete total words loaded = {:,}".format( grandtotal ) )

if debuggingThisModule:

for spaceCount in wordNumTotals:

print( " {} words: {}".format( spaceCount+1, wordNumTotals[spaceCount] ) )

editWindowObject.parentApp.setReadyStatus()

internalMarkers = None

DUMMY\_VALUE = 999999 # 设置一个很大数，最为初始最大值；

def countBookWords( BBB, internalBax, filename, isCurrentBook, internalMarkers ):

"""

找到所有关键字供用户选择使用

注意：该功能不能在内层中使用；

仅支持特定txt格式文件；

注意：交叉使用时，参数应该确切定义，确保值的唯一性；

"""

logging.debug( "countBookWords( {}, {}, {} )".format( BBB, internalBax, filename ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "countBookWords( {}, {}, {} )".format( BBB, internalBax, filename ) )

if BBB in AVOID\_BOOKS:

print( "Didn't load autocomplete words from {} {}".format( internalBax.getAName(), BBB ) )

return

countIncrement = 3 if isCurrentBook else 1

# 注意：如果失败，窗口将关闭

encoding = None

if encoding is None: encoding = 'utf-8'

lastLine, lineCount, lineDuples, lastMarker = '', 0, [], None

wordCounts = defaultdict( int )

def countWords( textLine ):

"""

打卡确认安全

"""

print( "countWords( {!r} )".format( textLine ) )

if '\\' in textLine:

print( " INT", marker, textLine )

for iMarker in internalMarkers:

print( " GOT", repr(iMarker) )

textLine = textLine.replace( iMarker+' ',' ' ).replace( iMarker+'\*',' ' )

if not '\\' in textLine: break

print( " NOW", marker, textLine )

words = textLine.replace('—','— ').replace('–','– ').split()

for wx,word in enumerate( words ):

if not word: continue

if 'XXX' in word: continue

singleWord = word

while singleWord and singleWord[-1] in END\_CHARS\_TO\_REMOVE:

singleWord = singleWord[:-1] # 去除特定按键

if len(singleWord) > 2: wordCounts[singleWord] += countIncrement

if word[-1] not in '—.–':

if wx < len(words)-1:

doubleWord = word+' '+words[wx+1]

print( 'doubleWord', repr(doubleWord) )

adjustedDoubleWord = doubleWord[:-1] if doubleWord[-1] in END\_CHARS\_TO\_REMOVE else doubleWord

if '. ' not in adjustedDoubleWord: #确定边界

wordCounts[adjustedDoubleWord] += countIncrement

if wx < len(words)-2:

tripleWord = doubleWord+' '+words[wx+2]

print( 'tripleWord', repr(tripleWord) )

adjustedTripleWord = tripleWord[:-1] if tripleWord[-1] in END\_CHARS\_TO\_REMOVE else tripleWord

if '. ' not in adjustedTripleWord:

wordCounts[adjustedTripleWord] += countIncrement

if wx < len(words)-3:

quadWord = tripleWord+' '+words[wx+3]

print( 'quadWord', repr(quadWord) )

adjustedQuadWord = quadWord[:-1] if quadWord[-1] in END\_CHARS\_TO\_REMOVE else quadWord

if '. ' not in adjustedQuadWord:

wordCounts[adjustedQuadWord] += countIncrement

if wx < len(words)-4:

quinWord = quadWord+' '+words[wx+4]

print( 'quinWord', repr(quinWord) )

adjustedQuinWord = quinWord[:-1] if quinWord[-1] in END\_CHARS\_TO\_REMOVE else quinWord

if '. ' not in adjustedQuinWord:

wordCounts[adjustedQuinWord] += countIncrement

USFMFilepath = os.path.join( internalBax.sourceFolder, filename )

with open( USFMFilepath, 'rt', encoding=internalBax.encoding ) as bookFile:

try:

for line in bookFile:

lineCount += 1

if lineCount==1 and encoding.lower()=='utf-8' and line[0]==chr(65279):

logging.info( "countBookWords: Detected Unicode Byte Order Marker (BOM) in {}".format( USFMFilepath ) )

line = line[1:]

if line and line[-1]=='\n': line=line[:-1]

if not line: continue

lastLine = line

print ( 'USFM file line is {!r}'.format( line ) )

if line[0:2]=='\\\_': continue

if line[0]=='#': continue

if line[0]!='\\':

if lastMarker is None:

logging.error( "countBookWords: Non-USFM line in {} -- line ignored at #{}".format( USFMFilepath, lineCount) )

print( "SFMFile.py: XXZXResult is", lineDuples, len(line) )

for x in range(0, min(6,len(line))):

print( x, "'" + str(ord(line[x])) + "'" )

raise IOError('Oops: Line break on last line ??? not handled here "' + line + '"')

else: # 追加继续线

if lastMarker in USFM\_PRINTABLE\_MARKERS:

oldmarker, oldtext = lineDuples.pop()

print ("Popped",oldmarker,oldtext)

print ("Adding", line, "to", oldmarker, oldtext)

lineDuples.append( (oldmarker, oldtext+' '+line) )

countWords( line )

continue

lineAfterBackslash = line[1:]

si1 = lineAfterBackslash.find( ' ' )

si2 = lineAfterBackslash.find( '\*' )

si3 = lineAfterBackslash.find( '\\' )

if si1==-1: si1 = DUMMY\_VALUE

if si2==-1: si2 = DUMMY\_VALUE

if si3==-1: si3 = DUMMY\_VALUE

si = min( si1, si2, si3 )

if si != DUMMY\_VALUE:

if si == si3:

marker = lineAfterBackslash[:si3]

text = lineAfterBackslash[si3:]

elif si == si2:

marker = lineAfterBackslash[:si2+1]

text = lineAfterBackslash[si2+1:]

elif si == si1:

marker = lineAfterBackslash[:si1]

text = lineAfterBackslash[si1+1:]

else:

marker = lineAfterBackslash

text = ''

print( " ", repr(marker), repr(text) )

if marker not in ignoreSFMs:

if marker in USFM\_PRINTABLE\_MARKERS and text:

print( " 1", marker, text )

if marker == 'v' and text[0].isdigit():

try: text = text.split( None, 1 )[1]

except IndexError: text = ''

print( " 2", marker, text )

countWords( text )

if not lineDuples: # 检测起始值

lineDuples.append( (marker, text) )

lastMarker = marker

except UnicodeError as err:

print( "Unicode error:", sys.exc\_info()[0], err )

logging.critical( "countBookWords: Invalid line in {} -- line ignored at #{}".format( USFMFilepath, lineCount) )

if lineCount > 1: print( 'Previous line was: ', lastLine )

print( line )

raise

return wordCounts

def countBookWordsHelper( parameters ):

"""

参数使用元胞数组格式，详细设置边界情况；

多进程运行时，边界将被自动设置；

"""

return countBookWords( \*parameters )

def loadBaxBookAutocompleteWords( editWindowObject ):

"""

加载所有额外设置；

对额外设置进行组合评估；

"""

logging.info( exp("loadBiookAutocompleteWords()") )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("loadBibkAutocompleteWords()") )

editWindowObject.parentApp.setDebugText( "loadBookAutocompleteWords…" )

editWindowObject.parentApp.setWaitStatus( \_("Loading {} Bax book words…").format( editWindowObject.projectName ) )

currentBBB = editWindowObject.currentVerseKey.getBBB()

print( " got BBB", repr(BBB) )

if currentBBB == 'UNK': return # 未识别

if not editWindowObject.internalBax.preloadDone: editWindowObject.internalBax.preload()

foundFilename = None

for BBB2,filename in editWindowObject.internalBax.maximumPossibleFilenameTuples:

if BBB2 == currentBBB: foundFilename = filename; break

wordCountResults = countBookWords( currentBBB, editWindowObject.internalBax, foundFilename, False )

print( 'wordCountResults', len(wordCountResults) )

Would be nice to load current book first, but we don't know it yet

autocompleteWords = []

if BaxOrgSysGlobals.debugFlag:

autocompleteWords = [ 'Lord God', 'Lord your(pl) God', '(is)', '(are)', '(were)', '(one who)', ]

try:

print( 'wordCountResults', currentBBB, discoveryResults )

print( currentBBB, 'mTWC', discoveryResults['mainTextWordCounts'] )

qqq = sorted( discoveryResults['mainTextWordCounts'].items(), key=lambda c: -c[1] )

print( 'qqq', qqq )

for word,count in sorted( wordCountResults.items(),

key=lambda duple: -duple[1] ):

if len(word) >= editWindowObject.autocompleteMinLength \

and word not in autocompleteWords:

if ' ' not in word or count > 4:

autocompleteWords.append( word )

else: print( 'loadBaxBookAutocompleteWords discarding', repr(word) )

except KeyError:

print( "Why did {} have no words???".format( currentBBB ) )

pass # Nothing for this book

print( 'autocompleteWords', len(autocompleteWords) )

setAutocompleteWords( editWindowObject, autocompleteWords )

editWindowObject.addAllNewWords = True

def loadBaxAutocompleteWords( editWindowObject ):

"""

加载用户设置；

"""

startTime = time.time()

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.loadBaxAutocompleteWords()") )

editWindowObject.parentApp.setDebugText( "loadBaxAutocompleteWords…" )

global internalMarkers

if internalMarkers is None:

internalMarkers = BaxOrgSysGlobals.USFMMarkers.getNoteMarkersList() \

+ BaxOrgSysGlobals.USFMMarkers.getCharacterMarkersList( includeBackslash=False, includeEndMarkers=False, includeNestedMarkers=True, expandNumberableMarkers=True )

internalMarkers = ['\\'+marker for marker in internalMarkers]

editWindowObject.parentApp.setWaitStatus( \_("Loading {} Bax words…").format( editWindowObject.projectName ) )

currentBBB = editWindowObject.currentVerseKey.getBBB()

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: print( " got current BBB", repr(currentBBB) )

if not editWindowObject.internalBax.preloadDone: editWindowObject.internalBax.preload()

bookWordCounts = {}

if editWindowObject.internalBax.maximumPossibleFilenameTuples:

if BaxOrgSysGlobals.maxProcesses > 1:

parameters = [(BBB,editWindowObject.internalBax,filename,BBB==currentBBB,internalMarkers) for BBB,filename in editWindowObject.internalBax.maximumPossibleFilenameTuples] # Can only pass a single parameter to map

if BaxOrgSysGlobals.verbosityLevel > 1:

print( exp("Autocomplete: loading up to {} USFM books using {} processes…").format( len(editWindowObject.internalBax.maximumPossibleFilenameTuples), BaxOrgSysGlobals.maxProcesses ) )

print( " NOTE: Outputs (including error & warning messages) from loading words from Bax books may be interspersed." )

BaxOrgSysGlobals.alreadyMultiprocessing = True

with multiprocessing.Pool( processes=BaxOrgSysGlobals.maxProcesses ) as pool:

results = pool.map( countBookWordsHelper, parameters )

assert len(results) == len(editWindowObject.internalBax.maximumPossibleFilenameTuples)

for (BBB,filename),counts in zip( editWindowObject.internalBax.maximumPossibleFilenameTuples, results ):

print( "XX", BBB, filename, len(counts) if counts else counts )

bookWordCounts[BBB] = counts

BaxOrgSysGlobals.alreadyMultiprocessing = False

else:

for BBB,filename in editWindowObject.internalBax.maximumPossibleFilenameTuples:

if BaxOrgSysGlobals.verbosityLevel>1 or BaxOrgSysGlobals.debugFlag:

print( \_(" USFMBax: Loading {} from {} from {}…").format( BBB, editWindowObject.internalBax.getAName(), editWindowObject.internalBax.sourceFolder ) )

bookWordCounts[BBB] = countBookWords( BBB, editWindowObject.internalBax, filename, BBB==currentBBB ) # also saves it

else:

logging.critical( "Autocomplete: " + \_("No books to load in folder '{}'!").format( editWindowObject.internalBax.sourceFolder ) )

autocompleteCounts = {}

for BBB,counts in bookWordCounts.items():

print( "here", BBB, len(counts) )

if counts:

for word, count in counts.items():

print( " ", word, count )

if len(word) >= editWindowObject.autocompleteMinLength:

if word in autocompleteCounts: autocompleteCounts[word] += count

else: autocompleteCounts[word] = count

print( "there", len(autocompleteCounts) )

autocompleteWords = []

if BaxOrgSysGlobals.debugFlag:

autocompleteWords = [ 'Lord God', 'Lord your(pl) God', '(is)', '(are)', '(were)', '(one who)', ]

for word,count in sorted( autocompleteCounts.items(), key=lambda duple: -duple[1] ):

if ' ' not in word or count > 9:

autocompleteWords.append( word )

else:

print( 'loadBaxAutocompleteWords discarding', repr(word) )

if ' ' not in word: halt

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: print( 'acW', autocompleteWords )

print( 'autocompleteWords', len(autocompleteWords) )

setAutocompleteWords( editWindowObject, autocompleteWords )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "loadBaxAutocompleteWords took", time.time()-startTime )

editWindowObject.addAllNewWords = True

def loadHunspellAutocompleteWords( editWindowObject, dictionaryFilepath, encoding='utf-8' ):

'''

界面菜单辅助信息设置

'''

logging.info( exp("loadHunspellAutocompleteWords( {}, {} )").format( dictionaryFilepath, encoding ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("loadHunspellAutocompleteWords( {}, {} )").format( dictionaryFilepath, encoding ) )

editWindowObject.parentApp.setDebugText( "loadHunspellAutocompleteWords…" )

editWindowObject.parentApp.setWaitStatus( \_("Loading dictionary…") )

internalCount = None

autocompleteWords = []

lineCount = 0

with open( dictionaryFilepath, 'rt', encoding=encoding ) as dictionaryFile:

for line in dictionaryFile:

lineCount += 1

if lineCount==1 and encoding.lower()=='utf-8' and line[0]==chr(65279):

logging.info( "loadHunspellAutocompleteWords: Detected Unicode Byte Order Marker (BOM) in {}".format( dictionaryFilepath ) )

line = line[1:]

if line and line[-1]=='\n': line=line[:-1]

if not line: continue

print( "line", lineCount, repr(line) )

if lineCount==1 and line.isdigit():

internalCount = int( line )

continue

try: word, codes = line.split( '/', 1 )

except ValueError: word, codes = line, ''

if word in ('3GPP','AA','ACAS',): continue #

print( "word", repr(word), repr(codes) )

autocompleteWords.append( word )

wordDeleteA = word[:-1] if word[-1]=='a' else word

wordDeleteE = word[:-1] if word[-1]=='e' else word

wordDeleteEY = word[:-1] if word[-1] in ('e','y',) else word

wordDeleteEChangeY = wordDeleteE[:-1]+'i' if wordDeleteE[-1]=='y' else wordDeleteE

wordAddEAfterSY = word+'e' if word[-1]=='s' else word

wordAddEAfterSY = wordAddEAfterSY[:-1]+'ie' if wordAddEAfterSY[-1]=='y' else wordAddEAfterSY

wordYtoI = word[:-1]+'i' if word[-1]=='y' else word

generatedWords = []

for code in codes:

print( " code", code, "for", repr(word) )

if code == 'A': generatedWords.append( 're' + word )

elif code == 'a': generatedWords.append( 'mis' + word )

elif code == 'B': generatedWords.append( word+'able' ); generatedWords.append( word+'ability' )

elif code == 'b': generatedWords.append( wordDeleteE + 'ible' ); generatedWords.append( wordDeleteE + 'ibility' )

elif code == 'C': generatedWords.append( 'de' + word )

elif code == 'c': generatedWords.append( 'over' + word )

elif code == 'D': generatedWords.append( wordDeleteE + 'ed' )

elif code == 'd': generatedWords.append( word+'ed' ); generatedWords.append( word+'ing' )

elif code == 'E': generatedWords.append( 'dis' + word )

elif code == 'e': generatedWords.append( 'out' + word )

elif code == 'F':

if word[0] in ( 'm','b','p',): generatedWords.append( 'com' + word )

elif word[0] == 'l': generatedWords.append( 'il' + word )

elif word[0] == 'r': generatedWords.append( 'ir' + word )

else: generatedWords.append( 'con' + word )

elif code == 'f': generatedWords.append( 'under' + word )

elif code == 'G':

generatedWords.append( wordDeleteE + 'ing' )

elif code == 'g':

generatedWords.append( wordDeleteE + 'ability' )

elif code == 'H':

generatedWords.append( word + 'th' ); generatedWords.append( word + 'fold' )

elif code == 'h':

generatedWords.append( wordDeleteE + 'edly' )

elif code == 'I':

if word[0] in ( 'm','b','p',): generatedWords.append( 'im' + word )

elif word[0] == 'l': generatedWords.append( 'il' + word )

elif word[0] == 'r': generatedWords.append( 'ir' + word )

else: generatedWords.append( 'in' + word )

elif code == 'i': generatedWords.append( wordDeleteEY + 'edness' )

elif code == 'J':

generatedWords.append( word + 'ings' )

elif code == 'j':

generatedWords.append( word + 'fully' )

elif code == 'K': generatedWords.append( 'pre' + word )

elif code == 'k': generatedWords.append( wordDeleteE + 'ingly' )

elif code == 'L': generatedWords.append( word+'ment' ); generatedWords.append( word+'ments' ); generatedWords.append( word+"ment's" )

elif code == 'l':

generatedWords.append( word + 'ably' )

elif code == 'M':

generatedWords.append( word + "'s" )

elif code == 'm':

generatedWords.append( word+'man' ); generatedWords.append( word+"man's" ); generatedWords.append( word+'men' ); generatedWords.append( word+"men's" )

elif code == 'N':

generatedWords.append( wordDeleteE + 'ion' )

elif code == 'n': generatedWords.append( wordDeleteE+'ion' ); generatedWords.append( wordDeleteE+'ions' )

elif code == 'O':

generatedWords.append( 'non' + word )

elif code == 'o':

generatedWords.append( wordDeleteA + 'ally' )

elif code == 'P':

generatedWords.append( wordYtoI+'ness' ); generatedWords.append( wordYtoI+"ness's" )

elif code == 'p':

generatedWords.append( word+'less' )

elif code == 'Q':

generatedWords.append( wordDeleteE+'ise' ); generatedWords.append( wordDeleteE+'ised' ); generatedWords.append( wordDeleteE+'ises' ); generatedWords.append( wordDeleteE+'ising' )

generatedWords.append( wordDeleteE+'ize' ); generatedWords.append( wordDeleteE+'ized' ); generatedWords.append( wordDeleteE+'izes' ); generatedWords.append( wordDeleteE+'izing' )

elif code == 'R':

generatedWords.append( wordDeleteE+'er' ); generatedWords.append( wordDeleteE+'ers' ); generatedWords.append( wordDeleteE+"er's" )

elif code == 'r':

generatedWords.append( word+'er' ); generatedWords.append( word+'ers' ); generatedWords.append( word+"er's" )

elif code == 'S': generatedWords.append( wordAddEAfterSY + 's' )

elif code == 'T': generatedWords.append( wordDeleteEY+'er' ); generatedWords.append( wordDeleteEY+'est' )

elif code == 'U': generatedWords.append( 'un' + word )

elif code == 'u': generatedWords.append( wordDeleteEY + 'iveness' )

elif code == 'V': generatedWords.append( wordDeleteEY + 'ive' )

elif code == 'v': generatedWords.append( wordDeleteEY + 'ively' )

elif code == 'W': generatedWords.append( wordDeleteEY + 'ic' )

elif code == 'w': generatedWords.append( wordDeleteEY + 'ical' )

elif code == 'X':

generatedWords.append( wordDeleteEY + 'ions' )

elif code == 'x': generatedWords.append( wordDeleteEY + 'ional' ); generatedWords.append( word + 'ionally' )

elif code == 'Y': generatedWords.append( wordYtoI + 'ly' )

elif code == 'y': generatedWords.append( word + 'ry' )

elif code == 'Z':

generatedWords.append( wordDeleteE + 'y' )

elif code == 'z':

generatedWords.append( wordDeleteEY + 'ily' )

elif code == '1': print( ' 1 on', repr(word), 'ignored' )

elif code == '2':

generatedWords.append( wordDeleteEY + 'iness' )

elif code == '3': generatedWords.append( wordDeleteEY+'ist' ); generatedWords.append( wordDeleteE+'ists' ); generatedWords.append( wordDeleteE+"ist's" )

elif code == '5':

generatedWords.append( word+'woman' ); generatedWords.append( word+"woman's" ); generatedWords.append( word+'women' ); generatedWords.append( word+"women's" )

elif code == '4': generatedWords.append( 'trans' + word )

elif code == '6':

generatedWords.append( word + 'ful' )

elif code == '7': generatedWords.append( word + 'able' )

elif BaxOrgSysGlobals.debugFlag:

print( lineCount, "code", code, "for", repr(word), repr(codes) )

halt

print( " generated", generatedWords, 'from', repr(word), repr(codes) )

autocompleteWords.extend( generatedWords )

lastLine = line

if lineCount > 60: break

print( 'acW', len(autocompleteWords), autocompleteWords )

if editWindowObject.autocompleteMinLength < 4:

print( "NOTE: Lengthened autocompleteMinLength from {} to {}".format( editWindowObject.autocompleteMinLength, 4 ) )

editWindowObject.autocompleteMinLength = 4

setAutocompleteWords( editWindowObject, autocompleteWords )

editWindowObject.addAllNewWords = False

def loadILEXAutocompleteWords( editWindowObject, dictionaryFilepath, lgCodes=None ):

"""

加载模型算法训练信息

"""

logging.info( exp("loadILEXAutocompleteWords( {}, {} )").format( dictionaryFilepath, lgCodes ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("loadILEXAutocompleteWords( {}, {} )").format( dictionaryFilepath, lgCodes ) )

editWindowObject.parentApp.setDebugText( "loadILEXAutocompleteWords…" )

editWindowObject.parentApp.setWaitStatus( \_("Loading dictionary…") )

autocompleteWords = []

lineCount = 0

with open( dictionaryFilepath, 'rt', encoding='utf-8' ) as dictionaryFile:

for line in dictionaryFile:

lineCount += 1

if lineCount==1:

if line[0]==chr(65279):

logging.info( "loadILEXAutocompleteWords1: Detected Unicode Byte Order Marker (BOM) in {}".format( dictionaryFilepath ) )

line = line[1:]

elif line[:3] == 'ï»¿':

logging.info( "loadILEXAutocompleteWords2: Detected Unicode Byte Order Marker (BOM) in {}".format( dictionaryFilepath ) )

line = line[3:]

if line and line[-1]=='\n': line=line[:-1]

if not line: continue

print( "line", lineCount, repr(line) )

if line.startswith( '\\wd ' ):

word = line[4:]

if '\*' in word and word[-2] == '\*' and word[-1].isdigit():

word = word[:-2]

elif line.startswith( '\\lg ' ):

lgCode = line[4:]

assert len(lgCode) == 3

elif line.startswith( '\\ps ' ):

POS = line[4:]

if lgCodes is None or lgCode in lgCodes:

if POS != 'x':

if word not in autocompleteWords:

autocompleteWords.append( word )

lastLine = line

if lineCount > 600: break

print( 'acW', len(autocompleteWords), autocompleteWords )

if editWindowObject.autocompleteMinLength < 4:

print( "NOTE: Lengthened autocompleteMinLength from {} to {}".format( editWindowObject.autocompleteMinLength, 4 ) )

editWindowObject.autocompleteMinLength = 4

setAutocompleteWords( editWindowObject, autocompleteWords )

editWindowObject.addAllNewWords = False

def getCharactersBeforeCursor( self, charCount=1 ):

"""

需要自动预测校准功能

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.getCharactersBeforeCursor( {} )").format( charCount ) )

previousText = self.textBox.get( tk.INSERT+'-{}c'.format( charCount ), tk.INSERT )

print( 'getCharactersBeforeCursor: returning previousText', repr(previousText) )

return previousText

def getWordCharactersBeforeCursor( self, maxCount=4 ):

"""

在线训练学习校准；

自动追溯；

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.getWordCharactersBeforeCursor( {} )").format( maxCount ) )

previousText = self.textBox.get( tk.INSERT+'-{}c'.format( maxCount ), tk.INSERT )

print( "previousText", repr(previousText) )

wordText = ''

for previousChar in reversed( previousText ):

if previousChar in self.autocompleteWordChars:

wordText = previousChar + wordText

else: break

print( 'getWordCharactersBeforeCursor: returning wordText', repr(wordText) )

return wordText

def getCharactersAndWordBeforeCursor( self, maxCount=4 ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.getCharactersAndWordBeforeCursor( {} )").format( maxCount ) )

previousText = self.textBox.get( tk.INSERT+'-{}c'.format( maxCount ), tk.INSERT )

print( "previousText", repr(previousText) )

delimiterCount = 0

wordText = ''

for previousChar in reversed( previousText ):

if previousChar in self.autocompleteWordChars:

wordText = previousChar + wordText

elif previousChar in BaxOrgSysGlobals.TRAILING\_WORD\_END\_CHARS+MULTIPLE\_SPACE\_SUBSTITUTE+TRAILING\_SPACE\_SUBSTITUTE:

if delimiterCount > 0: break

print( "Found delimiter {!r}".format( previousChar ) )

wordText = previousChar + wordText

delimiterCount += 1

print( 'getCharactersAndWordBeforeCursor: returning wordText', repr(wordText) )

return wordText

def getWordBeforeSpace( self, maxCount=20 ):

"""

追溯在训练；

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.getWordBeforeSpace( {} )").format( maxCount ) )

previousText = self.textBox.get( tk.INSERT+'-{}c'.format( maxCount ), tk.INSERT )

print( "previousText1", repr(previousText) )

assert previousText and previousText[-1] in BaxOrgSysGlobals.TRAILING\_WORD\_END\_CHARS+MULTIPLE\_SPACE\_SUBSTITUTE+TRAILING\_SPACE\_SUBSTITUTE

previousText = previousText[:-1]

print( "previousText2", repr(previousText) )

wordText = ''

if 1 or previousText and previousText[-1].isalpha():

for previousChar in reversed( previousText ):

if previousChar in self.autocompleteWordChars:

wordText = previousChar + wordText

else: break

print( 'getWordBeforeSpace: returning word Text', repr(wordText) )

return wordText

def acceptAutocompleteSelection( self, includeTrailingSpace=False ):

"""

用于自动训练过程

已备预测使用

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.acceptAutocompleteSelection( {} )").format( includeTrailingSpace ) )

assert self.autocompleteBox is not None

currentWord = self.autocompleteBox.get( tk.ACTIVE )

print( ' autocompleteBox currentWord', currentWord )

self.removeAutocompleteBox()

if self.autocompleteOverlap:

print( "Have {!r} with overlap {!r}".format( currentWord, self.autocompleteOverlap ) )

assert currentWord.startswith( self.autocompleteOverlap )

currentWord = currentWord[len(self.autocompleteOverlap):]

Autocomplete by inserting the rest of the selected word plus a space

NOTE: The user has to backspace over the space if they don't want it (e.g., to put a period)

NOTE: The box reappears with the current code if we don't append the space -- would need to add a flag

self.textBox.insert( tk.INSERT, currentWord[len(self.autocompleteOverlap):] \

+ (' ' if includeTrailingSpace else '') )

print( "acceptAutocompleteSelection for {!r}".format( currentWord ) )

addNewAutocompleteWord( self, currentWord )

# 设置首位置

firstLetter, remainder = currentWord[0], currentWord[1:]

self.autocompleteWords[firstLetter].remove( remainder )

self.autocompleteWords[firstLetter].insert( 0, remainder )

def addNewAutocompleteWord( self, possibleNewWord ):

"""

添加新的算法

复合型评估预测

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( exp("AutocompleteFunctions.addNewAutocompleteWord( {!r} )").format( possibleNewWord ) )

assert isinstance( possibleNewWord, str )

assert possibleNewWord

if ' ' in possibleNewWord:

remainder = possibleNewWord

while ' ' in remainder:

individualWord, remainder = remainder.split( None, 1 )

print( " word={!r}, remainder={!r}".format( individualWord, remainder ) )

print( "Recursive1 of {!r}".format( individualWord ) )

addNewAutocompleteWord( self, individualWord )

print( "Recursive2 of {!r}".format( remainder ) )

addNewAutocompleteWord( self, remainder )

while possibleNewWord and possibleNewWord[-1] in END\_CHARS\_TO\_REMOVE:

possibleNewWord = possibleNewWord[:-1]

if len( possibleNewWord ) > self.autocompleteMinLength:

print( "Adding new autocomplete word: {!r}".format( possibleNewWord ) )

firstLetter, remainder = possibleNewWord[0], possibleNewWord[1:]

try: self.autocompleteWords[firstLetter].remove( remainder )

except ValueError: pass

except KeyError:

self.autocompleteWords[firstLetter] = []

self.autocompleteWords[firstLetter].insert( 0, remainder )

def demo():

"""

demo展示；

"""

if BaxOrgSysGlobals.verbosityLevel > 0: print( ProgNameVersion )

if BaxOrgSysGlobals.verbosityLevel > 1: print( " Available CPU count =", multiprocessing.cpu\_count() )

if BaxOrgSysGlobals.debugFlag: print( exp("Running demo…") )

tkRootWindow = tk.Tk()

tkRootWindow.title( ProgNameVersion )

tkRootWindow.textBox = tk.Text( tkRootWindow )

uEW = AutocompleteFunctions( tkRootWindow, None )

tkRootWindow.mainloop()

KNOWN\_HTML\_TAGS = ('!DOCTYPE','html','head','meta','link','title','body','div',

'h1','h2','h3','p','li','a','span','table','tr','td','i','b','em','small')

NON\_FORMATTING\_TAGS = 'html','head','body','div','table','tr','td',

HTML\_REPLACEMENTS = ('&nbsp;',' '),('&lt;','<'),('&gt;','>'),('&amp;','&'),

TRAILING\_SPACE\_SUBSTITUTE = '⦻'

MULTIPLE\_SPACE\_SUBSTITUTE = '⧦'

DOUBLE\_SPACE\_SUBSTITUTE = MULTIPLE\_SPACE\_SUBSTITUTE + MULTIPLE\_SPACE\_SUBSTITUTE

CLEANUP\_LAST\_MULTIPLE\_SPACE = MULTIPLE\_SPACE\_SUBSTITUTE + ' '

TRAILING\_SPACE\_LINE = ' \n'

TRAILING\_SPACE\_LINE\_SUBSTITUTE = TRAILING\_SPACE\_SUBSTITUTE + '\n'

ALL\_POSSIBLE\_SPACE\_CHARS = ' ' + TRAILING\_SPACE\_SUBSTITUTE + MULTIPLE\_SPACE\_SUBSTITUTE

class BEntry( Entry ):

"""

数据收集处理；

数据清理；

"""

def \_\_init\_\_( self, \*args, \*\*kwargs ):

"""

初始化

"""

if BaxOrgSysGlobals.debugFlag:

print( "BEntry.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

Entry.\_\_init\_\_( self, \*args, \*\*kwargs )

CallbackAddon.\_\_init\_\_( self )

pass

class BCombobox( Combobox ):

"""

结果展示框

"""

def \_\_init\_\_( self, \*args, \*\*kwargs ):

if BaxOrgSysGlobals.debugFlag:

print( "BCombobox.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

Combobox.\_\_init\_\_( self, \*args, \*\*kwargs )

CallbackAddon.\_\_init\_\_( self )

pass

class BText( tk.Text ):

"""

文字标记

"""

def \_\_init\_\_(self, master, \*\*kw):

"""

初始化

"""

tk.apply( tk.Text.\_\_init\_\_, (self, master), kw )

self.bind( ... )

pass

class HTMLTextBox( BText ):

"""

算法扩展

"""

def \_\_init\_\_( self, \*args, \*\*kwargs ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HTMLTextBox.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

BText.\_\_init\_\_( self, \*args, \*\*kwargs )

standardFont = DEFAULT\_FONTNAME + ' 12'

smallFont = DEFAULT\_FONTNAME + ' 10'

self.styleDict = {

'i': { 'font':standardFont+' italic' },

'b': { 'font':standardFont+' bold' },

'em': { 'font':standardFont+' bold' },

'p\_i': { 'font':standardFont+' italic' },

'p\_b': { 'font':standardFont+' bold' },

'p\_em': { 'font':standardFont+' bold' },

'span': { 'foreground':'red', 'font':standardFont },

'li': { 'lmargin1':4, 'lmargin2':4, 'background':'pink', 'font':standardFont },

'a': { 'foreground':'blue', 'font':standardFont, 'underline':1 },

'small\_p': { 'background':'pink', 'font':smallFont, 'spacing1':'1' },

'small\_p\_pGeneratedNotice': { 'justify':tk.CENTER, 'background':'green', 'font':smallFont, 'spacing1':'1' },

'small\_p\_a': { 'foreground':'blue', 'font':smallFont, 'underline':1, 'spacing1':'1' },

'small\_p\_b': { 'background':'pink', 'font':smallFont+' bold', 'spacing1':'1' },

'p': { 'background':'pink', 'font':standardFont, 'spacing1':'1' },

'pGeneratedNotice': { 'justify':tk.CENTER, 'background':'green', 'font':smallFont, 'spacing1':'1' },

'p\_a': { 'foreground':'blue', 'font':standardFont, 'underline':1, 'spacing1':'1' },

'p\_span': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanGreekWord': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanHebrewWord': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanKJVUsage': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanStatus': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanSource': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanSource\_b': { 'foreground':'red', 'background':'pink', 'font':standardFont+' bold' },

'p\_spanSource\_span': { 'foreground':'red', 'background':'pink', 'font':standardFont, 'spacing1':'1' },

'p\_spanSource\_spanDef': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanSource\_spanHebrew': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanSource\_spanStrongs': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanMeaning': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_spanMeaning\_b': { 'foreground':'red', 'background':'pink', 'font':standardFont+' bold' },

'p\_spanMeaning\_spanDef': { 'foreground':'red', 'background':'pink', 'font':standardFont },

'p\_span\_b': { 'foreground':'red', 'background':'pink', 'font':standardFont+' bold' },

'p\_spanKJVUsage\_b': { 'foreground':'red', 'background':'pink', 'font':standardFont+' bold' },

'td\_a': { 'foreground':'blue', 'font':standardFont, 'underline':1 },

'h1\_td\_a': { 'foreground':'blue', 'font':standardFont, 'underline':1 },

'h1': { 'justify':tk.CENTER, 'foreground':'blue', 'font':DEFAULT\_FONTNAME+' 15', 'spacing1':'1', 'spacing3':'0.5' },

'h1\_a': { 'justify':tk.CENTER, 'foreground':'blue', 'font':DEFAULT\_FONTNAME+' 15', 'spacing1':'1', 'spacing3':'0.5', 'underline':1 },

'h1PageHeading': { 'justify':tk.CENTER, 'foreground':'blue', 'font':DEFAULT\_FONTNAME+' 15', 'spacing1':'1', 'spacing3':'0.5' },

'h2': { 'justify':tk.CENTER, 'foreground':'green', 'font':DEFAULT\_FONTNAME+' 14', 'spacing1':'0.8', 'spacing3':'0.3' },

'h3': { 'justify':tk.CENTER, 'foreground':'orange', 'font':DEFAULT\_FONTNAME+' 13', 'spacing1':'0.5', 'spacing3':'0.2' },

}

for tag,styleEntry in self.styleDict.items():

self.tag\_configure( tag, \*\*styleEntry )

background='yellow', font='helvetica 14 bold', relief=tk.RAISED

"background", "bgstipple", "borderwidth", "elide", "fgstipple",

"font", "foreground", "justify", "lmargin1", "lmargin2", "offset",

"overstrike", "relief", "rmargin", "spacing1", "spacing2", "spacing3",

"tabs", "tabstyle", "underline", and "wrap".

aTags = ('a','p\_a','small\_p\_a','h1\_a')

if debuggingThisModule:

for tag in self.styleDict:

if tag.endswith( '\_a' ): assert tag in aTags

for tag in aTags:

assert tag in self.styleDict

self.tag\_bind( tag, '<Button-1>', self.openHyperlink )

self.tag\_bind( tag, '<Enter>', self.overHyperlink )

self.tag\_bind( tag, '<Leave>', self.leaveHyperlink )

self.\_lastOverLink = None

def insert( self, point, iText ):

"""

插入大数据

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HTMLTextBox.insert( {}, {} )".format( repr(point), repr(iText) ) )

if point != tk.END:

logging.critical( "HTMLTextBox.insert " + \_("doesn't know how to insert at {}").format( repr(point) ) )

BText.insert( self, point, iText )

return

remainingText = iText.replace( '\n', ' ' )

remainingText = remainingText.replace( '<br>','\n' ).replace( '<br />','\n' ).replace( '<br/>','\n' )

while ' ' in remainingText: remainingText = remainingText.replace( ' ', ' ' )

currentFormatTags, currentHTMLTags = [], []

first = True

while remainingText:

try: print( " Remaining: {}".format( repr(remainingText) ) )

except UnicodeEncodeError: print( " Remaining: {}".format( len(remainingText) ) )

ix = remainingText.find( '<' )

if ix == -1:

BText.insert( self, point, remainingText, currentFormatTags )

remainingText = ""

else:

if ix > 0:

insertText = remainingText[:ix]

if HTMLTag and HTMLTag == 'title':

pass

elif insertText:

combinedFormats, lastTag, link = '', None, None

print( "cFT", currentFormatTags )

for tag in currentFormatTags:

if tag.startswith( 'a=' ):

tag, link = 'a', tag[2:]

print( "Got <a> link {}".format( repr(link) ) )

if tag != lastTag:

if combinedFormats: combinedFormats += '\_'

combinedFormats += tag

lastTag = tag

print( "combinedFormats", repr(combinedFormats) )

if combinedFormats and combinedFormats not in self.styleDict:

print( " Missing format:", repr(combinedFormats), "cFT", currentFormatTags, "cHT", currentHTMLTags )

try: print( " on", repr(remainingText[:ix]) )

except UnicodeEncodeError: pass

insertText = remainingText[:ix]

print( " Got format:", repr(combinedFormats), "cFT", currentFormatTags, "cHT", currentHTMLTags, repr(insertText) )

if 'Hebrew' in combinedFormats:

print( "Reversing", repr(insertText ) )

insertText = insertText[::-1]

for htmlChars, replacementChars in HTML\_REPLACEMENTS:

insertText = insertText.replace( htmlChars, replacementChars )

if link: print( "insertMarks", repr( (combinedFormats, 'href'+link,) if link else combinedFormats ) )

if link:

hypertag = 'href' + link

BText.insert( self, point, insertText, (combinedFormats, hypertag,) )

self.tag\_bind( hypertag, '<Enter>', self.overHyperlink )

self.tag\_bind( hypertag, '<Leave>', self.leaveHyperlink )

else: BText.insert( self, point, insertText, combinedFormats )

first = False

remainingText = remainingText[ix:]

try: print( " tag", repr(remainingText[:5]) )

except UnicodeEncodeError: print( " tag" )

ixEnd = remainingText.find( '>' )

ixNext = remainingText.find( '<', 1 )

print( "ixEnd", ixEnd, "ixNext", ixNext )

if ixEnd == -1 \

or (ixEnd!=-1 and ixNext!=-1 and ixEnd>ixNext):

logging.critical( "HTMLTextBox.insert: " + \_("Missing close bracket") )

BText.insert( self, point, remainingText, currentFormatTags )

remainingText = ""

break

fullHTMLTag = remainingText[1:ixEnd]

remainingText = remainingText[ixEnd+1:]

if remainingText:

try: print( "after marker", remainingText[0] )

except UnicodeEncodeError: pass

if not fullHTMLTag:

logging.critical( "HTMLTextBox.insert: " + \_("Unexpected empty HTML tags") )

continue

selfClosing = fullHTMLTag[-1] == '/'

if selfClosing:

fullHTMLTag = fullHTMLTag[:-1]

try: print( "fullHTMLTag", repr(fullHTMLTag), "self-closing" if selfClosing else "" )

except UnicodeEncodeError: pass

fullHTMLTagBits = []

insideDoubleQuotes = False

currentField = ""

for char in fullHTMLTag:

if char in (' ',) and not insideDoubleQuotes:

fullHTMLTagBits.append( currentField )

currentField = ""

else:

currentField += char

if char == '"': insideDoubleQuotes = not insideDoubleQuotes

if currentField: fullHTMLTagBits.append( currentField )

print( "{} got {}".format( repr(fullHTMLTag), fullHTMLTagBits ) )

HTMLTag = fullHTMLTagBits[0]

print( "HTMLTag", repr(HTMLTag) )

if HTMLTag and HTMLTag[0] == '/':

assert len(fullHTMLTagBits) == 1

assert not selfClosing

HTMLTag = HTMLTag[1:]

print( "Got HTML {} close tag".format( repr(HTMLTag) ) )

print( "cHT1", currentHTMLTags )

print( "cFT1", currentFormatTags )

if currentHTMLTags and HTMLTag == currentHTMLTags[-1]:

currentHTMLTags.pop()

if HTMLTag not in NON\_FORMATTING\_TAGS:

currentFormatTags.pop()

elif currentHTMLTags:

logging.critical( "HTMLTextBox.insert: " + \_("Expected to close {} but got {} instead").format( repr(currentHTMLTags[-1]), repr(HTMLTag) ) )

else:

logging.critical( "HTMLTextBox.insert: " + \_("Unexpected HTML close {} close marker").format( repr(HTMLTag) ) )

print( "cHT2", currentHTMLTags )

print( "cFT2", currentFormatTags )

else:

if HTMLTag not in KNOWN\_HTML\_TAGS:

logging.critical( \_("HTMLTextBox doesn't recognise or handle {} as an HTML tag").format( repr(HTMLTag) ) )

currentHTMLTags.append( HTMLTag )

continue

if HTMLTag in ('h1','h2','h3','p','li','table','tr',):

BText.insert( self, point, '\n' )

elif HTMLTag in ('li',):

BText.insert( self, point, '\n' )

elif HTMLTag in ('td',):

BText.insert( self, point, '\t' )

formatTag = HTMLTag

if len(fullHTMLTagBits)>1:

print( "Looking for attributes" )

for bit in fullHTMLTagBits[1:]:

try: print( " bit", repr(bit) )

except UnicodeEncodeError: pass

if bit.startswith('class="') and bit[-1]=='"':

formatTag += bit[7:-1]

elif formatTag=='a' and bit.startswith('href="') and bit[-1]=='"':

formatTag += '=' + bit[6:-1]

else: logging.error( "HTMLTextBox: " + \_("Ignoring {} attribute on {!r} tag").format( bit, HTMLTag ) )

if not selfClosing:

if HTMLTag != '!DOCTYPE':

currentHTMLTags.append( HTMLTag )

if HTMLTag not in NON\_FORMATTING\_TAGS:

currentFormatTags.append( formatTag )

if currentHTMLTags:

logging.critical( "HTMLTextBox.insert: " + \_("Left-over HTML tags: {}").format( currentHTMLTags ) )

if currentFormatTags:

logging.critical( "HTMLTextBox.insert: " + \_("Left-over format tags: {}").format( currentFormatTags ) )

def \_getURL( self, event ):

"""

确认数据位置

"""

xy = '@{0},{1}'.format( event.x, event.y )

print( "xy", repr(xy) )

print( "ixy", repr(self.index(xy)) )

tagNames = self.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'href' ):

URL = tagName[4:]

print( "URL", repr(URL) )

return URL

def openHyperlink( self, event ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: print( "HTMLTextBox.openHyperlink()" )

URL = self.\_getURL( event )

if BaxOrgSysGlobals.debugFlag:

xy = '@{0},{1}'.format( event.x, event.y )

tagNames = self.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'href' ): break

tag\_range = self.tag\_prevrange( tagName, xy )

print( "tr", repr(tag\_range) )

clickedText = self.get( \*tag\_range )

print( "Clicked on {}".format( repr(clickedText) ) )

if URL: self.master.gotoLink( URL )

def overHyperlink( self, event ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: print( "HTMLTextBox.overHyperlink()" )

URL = self.\_getURL( event )

if BaxOrgSysGlobals.debugFlag:

xy = '@{0},{1}'.format( event.x, event.y )

tagNames = self.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'href' ): break

tag\_range = self.tag\_prevrange( tagName, xy )

print( "tr", repr(tag\_range) )

clickedText = self.get( \*tag\_range )

print( "Over {}".format( repr(clickedText) ) )

if URL: self.master.overLink( URL )

def leaveHyperlink( self, event ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule: print( "HTMLTextBox.leaveHyperlink()" )

self.master.leaveLink()

class CallbackAddon():

"""

添加数据分布方法

"""

def \_\_init\_\_( self ):

if BaxOrgSysGlobals.debugFlag:

print( "CallbackAddon.\_\_init\_\_()" )

self.callbackFunction = None

private\_callback = self.register( self.\_callback )

self.tk.eval( """

rename {widget} \_{widget}

interp alias {{}} ::{widget} {{}} widget\_proxy \_{widget} {callback}

""".format( widget=str(self), callback=private\_callback ) )

self.autocorrectEntries = []

self.autocorrectEntries.append( ('<<','“') )

self.autocorrectEntries.append( ('“<','‘') )

self.autocorrectEntries.append( ('‘<',"'") )

self.autocorrectEntries.append( ("'<",'<') )

self.autocorrectEntries.append( ('>>','”') )

self.autocorrectEntries.append( ('”>','’') )

self.autocorrectEntries.append( ('’>',"'") )

self.autocorrectEntries.append( ("'>",'>') )

self.autocorrectEntries.append( ('--','–') )

self.autocorrectEntries.append( ('–-','—') )

self.autocorrectEntries.append( ('—-','-') )

self.autocorrectEntries.append( ('...','…') )

self.setTextChangeCallback( self.onTextChange )

def \_callback( self, result, \*args ):

"""

返回值

"""

if self.callbackFunction is not None:

self.callbackFunction( result, \*args )

def setTextChangeCallback( self, callableFunction ):

self.callbackFunction = callableFunction

def onTextChange( self, result, \*args ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "CallbackAddon.onTextChange( {}, {} )".format( repr(result), args ) )

if 0: # Get line and column info

lineColumn = self.index( tk.INSERT )

print( "lc", repr(lineColumn) )

line, column = lineColumn.split( '.', 1 )

print( "l,c", repr(line), repr(column) )

if 0:

tagNames = self.tag\_names( tk.INSERT )

tagNames2 = self.tag\_names( lineColumn )

tagNames3 = self.tag\_names( tk.INSERT + ' linestart' )

tagNames4 = self.tag\_names( lineColumn + ' linestart' )

tagNames5 = self.tag\_names( tk.INSERT + ' linestart+1c' )

tagNames6 = self.tag\_names( lineColumn + ' linestart+1c' )

print( "tN", tagNames )

if tagNames2!=tagNames or tagNames3!=tagNames or tagNames4!=tagNames or tagNames5!=tagNames or tagNames6!=tagNames:

print( "tN2", tagNames2 )

print( "tN3", tagNames3 )

print( "tN4", tagNames4 )

print( "tN5", tagNames5 )

print( "tN6", tagNames6 )

halt

if 0:

mark1 = self.mark\_previous( tk.INSERT )

mark2 = self.mark\_previous( lineColumn )

mark3 = self.mark\_previous( tk.INSERT + ' linestart' )

mark4 = self.mark\_previous( lineColumn + ' linestart' )

mark5 = self.mark\_previous( tk.INSERT + ' linestart+1c' )

mark6 = self.mark\_previous( lineColumn + ' linestart+1c' )

print( "mark1", mark1 )

if mark2!=mark1:

print( "mark2", mark1 )

if mark3!=mark1 or mark4!=mark1 or mark5!=mark1 or mark6!=mark1:

print( "mark3", mark3 )

if mark4!=mark3:

print( "mark4", mark4 )

print( "mark5", mark5 )

if mark6!=mark5:

print( "mark6", mark6 )

if self.autocorrectEntries and args[0]=='insert' and args[1]=='insert':

print( "Handle autocorrect" )

previousText = getCharactersBeforeCursor( self )

allText = self.get()

print( "allText", repr(allText) )

index = self.index( tk.INSERT )

print( "index", repr(index) )

previousText = allText[0:index]

print( "previousText", repr(previousText) )

for inChars,outChars in self.autocorrectEntries:

if previousText.endswith( inChars ):

print( "Going to replace {!r} with {!r}".format( inChars, outChars ) )

self.delete( index-len(inChars), index )

self.insert( tk.INSERT, outChars )

break

class CustomEntry( CallbackAddon, BEntry ):

def \_\_init\_\_( self, \*args, \*\*kwargs ):

if BaxOrgSysGlobals.debugFlag:

print( "CustomEntry.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

BEntry.\_\_init\_\_( self, \*args, \*\*kwargs ) # 初始化基础类

CallbackAddon.\_\_init\_\_( self )

class CustomCombobox( CallbackAddon, BCombobox ):

def \_\_init\_\_( self, \*args, \*\*kwargs ):

if BaxOrgSysGlobals.debugFlag:

print( "CustomCombobox.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

BCombobox.\_\_init\_\_( self, \*args, \*\*kwargs )

CallbackAddon.\_\_init\_\_( self )

class CustomText( CallbackAddon, BText ):

"""

通用文字描述

"""

def \_\_init\_\_( self, \*args, \*\*kwargs ):

if BaxOrgSysGlobals.debugFlag:

print( "CustomText.\_\_init\_\_( {}, {} )".format( args, kwargs ) )

BText.\_\_init\_\_( self, \*args, \*\*kwargs )

CallbackAddon.\_\_init\_\_( self )

def highlightPattern( self, pattern, styleTag, startAt=tkSTART, endAt=tk.END, regexpFlag=True ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "CustomText.highlightPattern( {}, {}, start={}, end={}, regexp={} )".format( pattern, styleTag, startAt, endAt, regexpFlag ) )

countVar = tk.IntVar()

matchEnd = startAt

while True:

print( "here0 mS={!r} mE={!r} sL={!r}".format( self.index("matchStart"), self.index("matchEnd"), self.index("searchLimit") ) )

index = self.search( pattern, matchEnd, stopindex=endAt, count=countVar, regexp=regexpFlag )

print( "here1", repr(index), repr(countVar.get()) )

if index == "": break

print( "here2", self.index("matchStart"), self.index("matchEnd") )

matchEnd = "{}+{}c".format( index, countVar.get() )

self.tag\_add( styleTag, index, matchEnd )

def highlightAllPatterns( self, patternCollection ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "CustomText.highlightAllPatterns( {} )".format( patternCollection ) )

for regexpFlag, pattern, tagName, tagDict in patternCollection:

self.tag\_configure( tagName, \*\*tagDict )

self.highlightPattern( pattern, tagName, regexpFlag=regexpFlag )

class ChildBoxAddon():

def \_\_init\_\_( self, parentWindow ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.\_\_init\_\_( {} )".format( parentWindow ) )

assert parentWindow

self.parentWindow = parentWindow

self.myKeyboardBindingsList = []

if BaxOrgSysGlobals.debugFlag: self.myKeyboardShortcutsList = []

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.\_\_init\_\_ finished." )

def \_createStandardBoxKeyboardBinding( self, name, commandFunction ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.\_createStandardBoxKeyboardBinding( {} )".format( name ) )

try: kBD = self.parentApp.keyBindingDict

except AttributeError:

kBD = self.parentWindow.parentApp.keyBindingDict

assert (name,kBD[name][0],) not in self.myKeyboardBindingsList

if name in kBD:

for keyCode in kBD[name][1:]:

print( "Bind {} for {}".format( repr(keyCode), repr(name) ) )

self.textBox.bind( keyCode, commandFunction )

if BaxOrgSysGlobals.debugFlag:

if keyCode in self.myKeyboardShortcutsList:

print( "ChildBoxAddon.\_createStandardBoxKeyboardBinding wants to add duplicate {}".format( keyCode ) )

self.myKeyboardShortcutsList.append( keyCode )

self.myKeyboardBindingsList.append( (name,kBD[name][0],) )

else: logging.critical( 'No key binding available for {}'.format( repr(name) ) )

def createStandardBoxKeyboardBindings( self, reset=False ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.createStandardBoxKeyboardBindings( {} )".format( reset ) )

if reset:

self.myKeyboardBindingsList = []

for name,commandFunction in ( ('SelectAll',self.doSelectAll),

('Find',self.doBoxFind), ('Refind',self.doBoxRefind),

('Help',self.doHelp), ('Info',self.doShowInfo), ('About',self.doAbout),

('ShowMain',self.parentWindow.doShowMainWindow),

('Close',self.doClose),

):

self.\_createStandardBoxKeyboardBinding( name, commandFunction )

def setFocus( self, event ):

"""

聚焦

"""

self.textBox.focus\_set()

def doCopy( self, event=None ):

"""

复制选中信息

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doCopy( {} )".format( event ) )

if not self.textBox.tag\_ranges( tk.SEL ):

errorBeep()

showError( self, APP\_NAME, \_("No text selected") )

else:

copyText = self.textBox.get( tk.SEL\_FIRST, tk.SEL\_LAST)

print( " copied text", repr(copyText) )

self.clipboard\_clear()

self.clipboard\_append( copyText )

def doSelectAll( self, event=None ):

"""

选择消息盒子里的信息

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doSelectAll( {} )".format( event ) )

self.textBox.tag\_add( tk.SEL, tkSTART, tk.END+'-1c' )

self.textBox.mark\_set( tk.INSERT, tkSTART )

self.textBox.see( tk.INSERT )

def doGotoWindowLine( self, event=None, forceline=None ):

self.parentApp.logUsage( ProgName, debuggingThisModule, 'ChildBoxAddon doGotoWindowLine {}'.format( forceline ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doGotoWindowLine( {}, {} )".format( event, forceline ) )

line = forceline or askinteger( APP\_NAME, \_("Enter line number") )

self.textBox.update()

self.textBox.focus()

if line is not None:

maxindex = self.textBox.index( tk.END+'-1c' )

maxline = int( maxindex.split('.')[0] )

if line > 0 and line <= maxline:

self.textBox.mark\_set( tk.INSERT, '{}.0'.format(line) )

self.textBox.tag\_remove( tk.SEL, tkSTART, tk.END )

self.textBox.tag\_add( tk.SEL, tk.INSERT, 'insert+1l' )

self.textBox.see( tk.INSERT )

else:

errorBeep()

showError( self, APP\_NAME, \_("No such line number") )

def doBoxFind( self, event=None, lastkey=None ):

self.parentApp.logUsage( ProgName, debuggingThisModule, 'ChildBoxAddon doBoxFind {!r}'.format( lastkey ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doBoxFind( {}, {!r} )".format( event, lastkey ) )

key = lastkey or askstring( APP\_NAME, \_("Enter search string"), parent=self )

self.textBox.update()

self.textBox.focus()

self.lastfind = key

if key:

nocase = self.optionsDict['caseinsens']

nocase = True

where = self.textBox.search( key, tkSTART if lastkey is None else tk.INSERT, tk.END, nocase=nocase )

if not where:

errorBeep()

showError( self, APP\_NAME, \_("String {!r} not found").format( key if len(key)<20 else (key[:18]+'…') ) )

else:

pastkey = where + '+%dc' % len(key)

self.textBox.tag\_remove( tk.SEL, tkSTART, tk.END )

self.textBox.tag\_add( tk.SEL, where, pastkey )

self.textBox.mark\_set( tk.INSERT, pastkey )

self.textBox.see( where )

def doBoxRefind( self, event=None ):

self.parentApp.logUsage( ProgName, debuggingThisModule, 'ChildBoxAddon doBoxRefind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doBoxRefind( {} ) for {!r}".format( event, self.lastfind ) )

self.doBoxFind( lastkey=self.lastfind )

def doShowInfo( self, event=None ):

"""

弹出对话框

"""

self.parentApp.logUsage( ProgName, debuggingThisModule, 'ChildBoxAddon doShowInfo' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doShowInfo( {} )".format( event ) )

text = self.getAllText()

numChars = len( text )

numLines = len( text.split('\n') )

numWords = len( text.split() )

index = self.textBox.index( tk.INSERT )

atLine, atColumn = index.split('.')

infoString = 'Current location:\n' \

+ ' Line:\t{}\n Column:\t{}\n'.format( atLine, atColumn ) \

+ '\nFile text statistics:\n' \

+ ' Chars:\t{}\n Lines:\t{}\n Words:\t{}'.format( numChars, numLines, numWords )

showInfo( self, 'Window Information', infoString )

def clearText( self ):

self.textBox.configure( state=tk.NORMAL )

self.textBox.delete( tkSTART, tk.END )

def isEmpty( self ):

return not self.getAllText()

def modified( self ):

"""

信息框修改监测

"""

print( "Configure", self.textBox.configure() )

print( " State", self.textBox.configure()['state'] )

if self.textBox.configure()['state'][4] == 'normal':

return self.textBox.edit\_modified()

else: return False

def getAllText( self ):

return self.textBox.get( tkSTART, tk.END+'-1c' )

def setAllText( self, newText ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.setAllText( {!r} )".format( newText ) )

self.textBox.configure( state=tk.NORMAL )

self.textBox.delete( tkSTART, tk.END )

self.textBox.insert( tk.END, newText )

self.textBox.mark\_set( tk.INSERT, tkSTART )

self.textBox.see( tk.INSERT )

self.textBox.edit\_reset()

self.textBox.edit\_modified( tk.FALSE )

def doShowMainWindow( self, event=None ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doShowMainWindow( {} )".format( event ) )

self.parentApp.rootWindow.iconify()

self.parentApp.rootWindow.withdraw()

self.parentApp.rootWindow.update()

self.parentApp.rootWindow.deiconify()

self.parentApp.rootWindow.focus\_set()

self.parentApp.rootWindow.lift()

def doClose( self, event=None ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "ChildBoxAddon.doClose( {} )".format( event ) )

self.destroy()

class BaxBoxAddon():

def \_\_init\_\_( self, parentWindow, BaxBoxType ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.\_\_init\_\_( {}, {} )".format( parentWindow, BaxBoxType ) )

assert parentWindow

self.parentWindow, self.BaxBoxType = parentWindow, BaxBoxType

for USFMKey, styleDict in self.parentWindow.parentApp.stylesheet.getTKStyles().items():

self.textBox.tag\_configure( USFMKey, \*\*styleDict )

self.textBox.tag\_configure( 'contextHeader', background='pink', font='helvetica 6 bold' )

self.textBox.tag\_configure( 'context', background='pink', font='helvetica 6' )

self.textBox.tag\_configure( 'markersHeader', background='yellow3', font='helvetica 6 bold' )

self.textBox.tag\_configure( 'markers', background='yellow3', font='helvetica 6' )

else:

self.textBox.tag\_configure( 'verseNumberFormat', foreground='blue', font='helvetica 8', relief=tk.RAISED, offset='3' )

self.textBox.tag\_configure( 'versePreSpaceFormat', background='pink', font='helvetica 8' )

self.textBox.tag\_configure( 'versePostSpaceFormat', background='pink', font='helvetica 4' )

self.textBox.tag\_configure( 'verseTextFormat', font='sil-doulos 12' )

self.textBox.tag\_configure( 'otherVerseTextFormat', font='sil-doulos 9' )

self.textBox.tag\_configure( 'verseText', background='yellow', font='helvetica 14 bold', relief=tk.RAISED )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.\_\_init\_\_ finished." )

def createStandardBoxKeyboardBindings( self, reset=False ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.createStandardBoxKeyboardBindings( {} )".format( reset ) )

if reset:

self.myKeyboardBindingsList = []

for name,commandFunction in ( ('SelectAll',self.doSelectAll),

('Find',self.doBaxFind),

('Help',self.doHelp), ('Info',self.doShowInfo), ('About',self.doAbout),

('ShowMain',self.doShowMainWindow),

('Close',self.doClose), ):

self.\_createStandardBoxKeyboardBinding( name, commandFunction )

class App(tk.Frame):

def \_\_init\_\_(self, master):

tk.Frame.\_\_init\_\_(self, master)

self.pack()

self.master.title("start")#开始启动

self.master.resizable(False, False)

self.master.tk\_setPalette(background='#execute')

x = (self.master.winfo\_screenwidth() - self.master.winfo\_reqwidth()) / 2

y = (self.master.winfo\_screenheight() - self.master.winfo\_reqheight()) / 3

self.master.geometry("+{}+{}".format(x, y))

self.master.config(menu=tk.Menu(self.master))

tk.Label(self, text="This is your first GUI. (highfive)").pack()

tk.Button(self, text='OK', default='active', command=self.click\_ok).pack(side='right')

tk.Button(self, text='Cancel', command=self.click\_cancel).pack(side='right')

def click\_ok(self):

print("The user clicked 'OK'") #点击ok键

def click\_cancel(self):

print("The user clicked 'Cancel'")#点击cancel键

self.master.destroy()

class App(tk.Frame):

def \_\_init\_\_(self, master):

tk.Frame.\_\_init\_\_(self, master)

self.pack()

self.master.title("welecome")

self.master.resizable(False, False)

self.master.tk\_setPalette(background='#ececec')

x = (self.master.winfo\_screenwidth() - self.master.winfo\_reqwidth()) / 2

y = (self.master.winfo\_screenheight() - self.master.winfo\_reqheight()) / 3

self.master.geometry("+{}+{}".format(x, y))

self.master.config(menu=tk.Menu(self.master))

dialog\_frame = tk.Frame(self)

dialog\_frame.pack(padx=20, pady=15)

tk.Label(dialog\_frame, text="This is your first GUI. (highfive)").pack()

button\_frame = tk.Frame(self)

button\_frame.pack(padx=15, pady=(0, 15), anchor='e')

tk.Button(button\_frame, text='OK', default='active', command=self.click\_ok).pack(side='right')

tk.Button(button\_frame, text='Cancel', command=self.click\_cancel).pack(side='right')

def click\_ok(self):

print("The user clicked 'OK'")#用户点击Ok键

def click\_cancel(self):

print("The user clicked 'Cancel'")#用户点击cancel键

self.master.destroy()

class App(tk.Frame):

def \_\_init\_\_(self, master):

tk.Frame.\_\_init\_\_(self, master)

self.pack()

self.master.title("")

self.master.resizable(False, False)

self.master.tk\_setPalette(background='#ececec')

self.master.protocol('WM\_DELETE\_WINDOW', self.click\_cancel)

self.master.bind('<Return>', self.click\_ok)

self.master.bind('<Escape>', self.click\_cancel)

x = (self.master.winfo\_screenwidth() - self.master.winfo\_reqwidth()) / 2

y = (self.master.winfo\_screenheight() - self.master.winfo\_reqheight()) / 3

self.master.geometry("+{}+{}".format(x, y))

self.master.config(menu=tk.Menu(self))

tk.Message(self, text="Please authenticate with your username and password before continuing.",

font='System 14 bold', justify='left', aspect=800).pack(pady=(15, 0))

dialog\_frame = tk.Frame(self)

dialog\_frame.pack(padx=20, pady=15, anchor='w')

tk.Label(dialog\_frame, text='Username:').grid(row=0, column=0, sticky='w')

self.user\_input = tk.Entry(dialog\_frame, background='white', width=24)

self.user\_input.grid(row=0, column=1, sticky='w')

self.user\_input.focus\_set()

tk.Label(dialog\_frame, text='Password:').grid(row=1, column=0, sticky='w')

self.pass\_input = tk.Entry(dialog\_frame, background='white', width=24, show='\*')

self.pass\_input.grid(row=1, column=1, sticky='w')

button\_frame = tk.Frame(self)

button\_frame.pack(padx=15, pady=(0, 15), anchor='e')

tk.Button(button\_frame, text='OK', height=1, width=6, default='active', command=self.click\_ok).pack(side='right')

tk.Button(button\_frame, text='Cancel', height=1, width=6, command=self.click\_cancel).pack(side='right', padx=10)

def click\_ok(self, event=None):

print("The user clicked 'OK':\nUsername: {}\nPassword: {}".format(self.user\_input.get(), self.pass\_input.get()))

self.master.destroy()

def click\_cancel(self, event=None):

print("The user clicked 'Cancel'")

self.master.destroy()

class App(tk.Frame):

def \_\_init\_\_(self, master):

tk.Frame.\_\_init\_\_(self, master)

self.pack()

self.master.title("Hello World")

self.master.resizable(False, False)

self.master.tk\_setPalette(background='#ececec')

x = (self.master.winfo\_screenwidth() - self.master.winfo\_reqwidth()) / 2

y = (self.master.winfo\_screenheight() - self.master.winfo\_reqheight()) / 3

self.master.geometry("+{}+{}".format(x, y))

self.master.config(menu=tk.Menu(self))

tk.Label(self, text="This is your first GUI. (highfive)").pack()

def kmeanTestAll(self,Xraw,Reraw,figStart): # 启动Kmean算法

Xshape=Xraw.shape

Xrow=Xshape[0]

Xcol=Xshape[1]

if figStart!=Xcol:

Xraw,X0,Reraw,y0=train\_test\_split(Xraw,np.array(Reraw),test\_size=0.0)

dispersity=[] #数据分散度

profitP=[]

if figStart!=0:

colStart=Xcol

else:

colStart=0

for lp in range(colStart,Xcol+1):

if lp<Xcol:

X=Xraw[:,lp]

figTitle=str(lp)

else:

X=Xraw

figTitle='All'

trainSample=30000

if Xrow<trainSample:

Xtrain=X[:Xrow//2]

Xtest=X[Xrow//2:]

Retrain=Reraw[:Xrow//2]

Retest=Reraw[Xrow//2:]

else:

Xtrain=X[:trainSample]

Xtest=X[trainSample:]

Retrain=Reraw[:trainSample]

Retest=Reraw[trainSample:]

if figStart!=0:

Xtest=X

Retest=Reraw

figTitle=str(figStart)

kmean=KMeans(n\_clusters=5).fit(np.row\_stack(Xtrain))

joblib.dump(kmean,self.saveData+figTitle+'\_kmean')

records=[]

for i in range(2):

if i==0:

Xtem=Xtrain

Retem=Retrain

else:

Xtem=Xtest

Retem=Retest

flag=kmean.predict(np.row\_stack(Xtem))

plt.figure(figsize=(15,8))

xi=[]

yi=[]

recordi=[]

for i2 in range(kmean.n\_clusters):

state=(flag==i2)

ReT=Retem[state]

ReTcs=ReT.cumsum()

LT=len(ReT)

if LT<2:

continue

maxDraw=0

maxDrawi=0

maxDrawValue=0

i2High=0

for i3 in range(LT):

if ReTcs[i3]>i2High:

i2High=ReTcs[i3]

drawT=i2High-ReTcs[i3]

if maxDraw<drawT:

maxDraw=drawT

maxDrawi=i3

maxDrawValue=ReTcs[i3]

xi.append(maxDrawi)

yi.append(maxDrawValue)

recordi.append([LT,np.mean(ReT)/np.std(ReT),ReTcs[-1]/LT\*100])

try:

plt.plot(range(LT),ReTcs,label='latent\_state %d;orders:%d;IR:%.4f;winratio(ratioWL):%.2f%%(%.2f);maxDraw:%.2f%%;profitP:%.4f%%;'\

%(i2,LT,np.mean(ReT)/np.std(ReT),sum(ReT>0)/float(LT),np.mean(ReT[ReT>0])/-np.mean(ReT[ReT<0]),maxDraw\*100,ReTcs[-1]/LT\*100))

except:

plt.plot(range(LT),ReTcs,label='latent\_state %d;orders:%d;IR:%.4f;winratio(ratioWL):%.2f%%(%s);maxDraw:%.2f%%;profitP:%.4f%%;'\

%(i2,LT,np.mean(ReT)/np.std(ReT),sum(ReT>0)/float(LT),'error',maxDraw\*100,ReTcs[-1]/LT\*100))

records.append(recordi)

plt.plot(xi,yi,'r\*')

plt.title(figTitle,fontsize=16)

if i==1:

if lp<Xcol:

tem=np.sort(Xtem)

pointsTem=tem[[list(map(int,np.linspace(0,len(tem),6)))[1:-1]]]

tem=np.array([np.mean(Retem[Xtem<=pointsTem[0]]),np.mean(Retem[(Xtem>pointsTem[0]) \* (Xtem<=pointsTem[1])]),np.mean(Retem[(Xtem>pointsTem[1])\*(Xtem<=pointsTem[2])]),\

np.mean(Retem[(Xtem>pointsTem[2])\*(Xtem<=pointsTem[3])]),np.mean(Retem[Xtem>pointsTem[3]])])

tem=(tem/max(tem)).std()

else:

tem=0

rec1=np.row\_stack(records[0])

rec2=np.row\_stack(records[1])

profitP.append(rec2[:,2].tolist())

dispersity.append(tem)

if tem>0.2:

plt.xlabel( 'indicator column %d, correlative of train and test: %.10f, dispesity:%.10f'\

%(lp,pd.DataFrame(rec1[:,1])[0].corr(pd.DataFrame(rec2[:,1])[0]), tem ),color='r')

else:

plt.xlabel( 'indicator column %d, correlative of train and test: %.10f, dispesity:%.10f'\

**项目后1500行**

print( "BaxBoxAddon.createContextMenu()" )

self.textBox.contextMenu = tk.Menu( self, tearoff=0 )

self.textBox.contextMenu.add\_command( label=\_('Copy'), underline=0, command=self.doCopy, accelerator=self.parentApp.keyBindingDict[\_('Copy')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Select all'), underline=7, command=self.doSelectAll, accelerator=self.parentApp.keyBindingDict[\_('SelectAll')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Bax Find…'), underline=6, command=self.doBaxFind, accelerator=self.parentApp.keyBindingDict[\_('Find')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Find in window…'), underline=8, command=self.doBoxFind )

self.contextMenu.add\_separator()

self.contextMenu.add\_command( label=\_('Close window'), underline=1, command=self.doClose, accelerator=self.parentApp.keyBindingDict[\_('Close')][0] )

self.textBox.bind( '<Button-3>', self.showContextMenu )

self.pack()

self.BaxFindOptionsDict, self.BaxReplaceOptionsDict = {}, {}

def showContextMenu( self, event ):

self.textBox.contextMenu.tk\_popup( event.x\_root, event.y\_root )

def displayAppendVerse( self, firstFlag, verseKey, verseContextData, lastFlag=True, currentVerseFlag=False, substituteTrailingSpaces=False, substituteMultipleSpaces=False ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "displayAppendVerse( {}, {}, {}, {}, {}, {}, {} )".format( firstFlag, verseKey, verseContextData, lastFlag, currentVerseFlag, substituteTrailingSpaces, substituteMultipleSpaces ) )

assert isinstance( firstFlag, bool )

assert isinstance( verseKey, SimpleVerseKey )

if verseContextData:

assert isinstance( verseContextData, tuple ) and len(verseContextData)==2 or isinstance( verseContextData, str )

assert isinstance( lastFlag, bool )

assert isinstance( currentVerseFlag, bool )

def insertAtEnd( ieText, ieTags ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "insertAtEnd( {!r}, {} )".format( ieText, ieTags ) )

assert isinstance( ieText, str )

assert isinstance( ieTags, (str,tuple) )

assert TRAILING\_SPACE\_SUBSTITUTE not in ieText

assert MULTIPLE\_SPACE\_SUBSTITUTE not in ieText

if substituteMultipleSpaces:

ieText = ieText.replace( ' ', DOUBLE\_SPACE\_SUBSTITUTE )

ieText = ieText.replace( CLEANUP\_LAST\_MULTIPLE\_SPACE, DOUBLE\_SPACE\_SUBSTITUTE )

if substituteTrailingSpaces:

ieText = ieText.replace( TRAILING\_SPACE\_LINE, TRAILING\_SPACE\_LINE\_SUBSTITUTE )

self.textBox.insert( tk.END, ieText, ieTags )

try: cVM, fVM = self.\_contextViewMode, self.\_formatViewMode

except AttributeError:

cVM, fVM = self.parentWindow.\_contextViewMode, self.parentWindow.\_formatViewMode

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "displayAppendVerse2( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.displayAppendVerse( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

try: print( "BaxBoxAddon.displayAppendVerse( {}, {}, {}, {} )".format( firstFlag, verseKey, verseContextData, currentVerseFlag ) )

except UnicodeEncodeError: print( "BaxBoxAddon.displayAppendVerse", firstFlag, verseKey, currentVerseFlag )

BBB, C, V = verseKey.getBCV()

C, V = int(C), int(V)

C1 = C2 = int(C); V1 = V2 = int(V)

if V1 > 0: V1 -= 1

elif C1 > 0:

C1 -= 1

V1 = self.getNumVerses( BBB, C1 )

if V2 < self.getNumVerses( BBB, C2 ): V2 += 1

elif C2 < self.getNumChapters( BBB):

C2 += 1

V2 = 0

previousMarkName = 'C{}V{}'.format( C1, V1 )

currentMarkName = 'C{}V{}'.format( C, V )

nextMarkName = 'C{}V{}'.format( C2, V2 )

print( "Marks", previousMarkName, currentMarkName, nextMarkName )

lastCharWasSpace = haveTextFlag = not firstFlag

if verseContextData is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "BaxBoxAddon.displayAppendVerse has no data for", verseKey )

verseDataList = context = None

elif isinstance( verseContextData, tuple ):

assert len(verseContextData) == 2

verseDataList, context = verseContextData

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " VerseDataList: {}".format( verseDataList ) )

print( " Context: {}".format( context ) )

elif isinstance( verseContextData, str ):

verseDataList, context = verseContextData.split( '\n' ), None

elif BaxOrgSysGlobals.debugFlag: halt

if firstFlag:

if context:

print( "context", context )

print( " Setting context mark to {}".format( previousMarkName ) )

self.textBox.mark\_set( previousMarkName, tk.INSERT )

self.textBox.mark\_gravity( previousMarkName, tk.LEFT )

insertAtEnd( ' '+\_("Prior context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in context:

print( " someMarker", someMarker )

if someMarker != 'chapters':

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

haveTextFlag = True

if verseDataList and fVM == 'Formatted':

firstEntry = verseDataList[0]

if isinstance( firstEntry, InternalBaxEntry ): marker = firstEntry.getMarker()

elif isinstance( firstEntry, tuple ): marker = firstEntry[0]

else: marker = None

if marker in BaxOrgSysGlobals.USFMParagraphMarkers:

insertAtEnd( ' '+\_("Current context")+': ', 'contextHeader' )

insertAtEnd( marker+' ', 'context' )

Display all line markers in this segment

markerList = []

for verseData in verseDataList:

if isinstance( verseData, InternalBaxEntry ): marker = verseData.getMarker()

elif isinstance( verseData, tuple ): marker = verseData[0]

else: marker = None

if marker and not marker.startswith('¬') \

and not marker.endswith('~') and not marker.endswith('#'):

markerList.append( marker )

if markerList:

insertAtEnd( ' '+\_("Displayed markers")+': ', 'markersHeader' )

insertAtEnd( str(markerList)[1:-1], 'markers' )

print( " Setting mark to {}".format( currentMarkName ) )

self.textBox.mark\_set( currentMarkName, tk.INSERT )

self.textBox.mark\_gravity( currentMarkName, tk.LEFT )

if verseDataList is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "BaxBoxAddon.displayAppendVerse has no data for", self.moduleID, verseKey )

self.textBox.insert( tk.END, '--' )

else:

hadVerseText = False

try: cVM = self.\_contextViewMode

except AttributeError: cVM = self.parentWindow.\_contextViewMode

lastParagraphMarker = context[-1] if context and context[-1] in BaxOrgSysGlobals.USFMParagraphMarkers \

else 'v~'

endMarkers = []

for verseDataEntry in verseDataList:

if isinstance( verseDataEntry, InternalBaxEntry ):

marker, cleanText = verseDataEntry.getMarker(), verseDataEntry.getCleanText()

elif isinstance( verseDataEntry, tuple ):

marker, cleanText = verseDataEntry[0], verseDataEntry[3]

elif isinstance( verseDataEntry, str ):

if verseDataEntry=='': continue

verseDataEntry += '\n'

if verseDataEntry[0]=='\\':

marker = ''

for char in verseDataEntry[1:]:

if char!='¬' and not char.isalnum(): break

marker += char

cleanText = verseDataEntry[len(marker)+1:].lstrip()

else:

marker, cleanText = None, verseDataEntry

elif BaxOrgSysGlobals.debugFlag: halt

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " displayAppendVerse", lastParagraphMarker, haveTextFlag, marker, repr(cleanText) )

if fVM == 'Unformatted':

if marker and marker[0]=='¬': pass

elif marker in ('intro','chapters','list',): pass

else:

if isinstance( verseDataEntry, str ):

print( "marker={!r}, verseDataEntry={!r}".format( marker, verseDataEntry ) )

insertAtEnd( verseDataEntry, marker )

else:

if hadVerseText and marker in ( 's', 's1', 's2', 's3' ):

print( " Setting s mark to {}".format( nextMarkName ) )

self.textBox.mark\_set( nextMarkName, tk.INSERT )

self.textBox.mark\_gravity( nextMarkName, tk.LEFT )

print( " Inserting ({}): {!r}".format( marker, verseDataEntry ) )

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

if marker is None:

insertAtEnd( cleanText, '###' )

else: insertAtEnd( '\\{} {}'.format( marker, cleanText ), marker+'#' )

haveTextFlag = True

elif fVM == 'Formatted':

if marker.startswith( '¬' ):

if marker != '¬v': endMarkers.append( marker )

else: endMarkers = []

if marker.startswith( '¬' ):

pass

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker == 'id':

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ide','rem',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('h','toc1','toc2','toc3','cl¤',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('intro','chapters','list',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('mt1','mt2','mt3','mt4', 'imt1','imt2','imt3','imt4', 'iot','io1','io2','io3','io4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ip','ipi','im','imi','ipq','imq','ipr', 'iq1','iq2','iq3','iq4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('s1','s2','s3','s4', 'is1','is2','is3','is4', 'ms1','ms2','ms3','ms4', 'cl',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('d','sp',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('r','mr','sr',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in BaxOrgSysGlobals.USFMParagraphMarkers:

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

lastParagraphMarker = marker

haveTextFlag = True

elif marker in ('b','ib'):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

elif marker in ('m','im'):

self.textBox.insert ( tk.END, '\n' if haveTextFlag else ' ', marker )

if cleanText:

insertAtEnd( cleanText, '\*'+marker if currentVerseFlag else marker )

lastCharWasSpace = False

haveTextFlag = True

elif marker == 'p#':

assert self.BaxBoxType in ('DBPBaxResourceBox','DBPBaxResourceWindow')

pass

elif marker == 'c':

if not firstFlag: haveC = cleanText

else: print( " Ignore C={}".format( cleanText ) )

pass

elif marker == 'c#':

if cleanText != verseKey.getBBB():

if not lastCharWasSpace: insertAtEnd( ' ', 'v-' )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

lastCharWasSpace = False

elif marker == 'v':

if cleanText != '1':

if haveTextFlag:

insertAtEnd( ' ', (lastParagraphMarker,'v-',) if lastParagraphMarker else ('v-',) )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

insertAtEnd( '\u2009', (lastParagraphMarker,'v+',) if lastParagraphMarker else ('v+',) )

lastCharWasSpace = haveTextFlag = True

elif marker in ('v~','p~'):

insertAtEnd( cleanText, '\*'+lastParagraphMarker if currentVerseFlag else lastParagraphMarker )

haveTextFlag = True

else:

if BaxOrgSysGlobals.debugFlag:

logging.critical( \_("BaxBoxAddon.displayAppendVerse (formatted): Unknown marker {!r} {!r} from {}").format( marker, cleanText, verseDataList ) )

else:

logging.critical( \_("BaxBoxAddon.displayAppendVerse (formatted): Unknown marker {!r} {!r}").format( marker, cleanText ) )

else:

logging.critical( \_("BaxBoxAddon.displayAppendVerse: Unknown {!r} format view mode").format( fVM ) )

if BaxOrgSysGlobals.debugFlag: halt

if lastFlag and cVM=='ByVerse' and endMarkers:

print( "endMarkers", endMarkers )

insertAtEnd( ' '+ \_("End context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in endMarkers:

print( " someMarker", someMarker )

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

def getBeforeAndAfterBaxData( self, newVerseKey ):

if BaxOrgSysGlobals.debugFlag:

print( "BaxBoxAddon.getBeforeAndAfterBaxData( {} )".format( newVerseKey ) )

assert isinstance( newVerseKey, SimpleVerseKey )

BBB, C, V = newVerseKey.getBCV()

intC, intV = newVerseKey.getChapterNumberInt(), newVerseKey.getVerseNumberInt()

prevBBB, prevIntC, prevIntV = BBB, intC, intV

previousVersesData = []

for n in range( -self.parentApp.viewVersesBefore, 0 ):

failed = False

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " getBeforeAndAfterBaxData here with", repr(n), repr(prevIntC), repr(prevIntV) )

if prevIntV is not None and prevIntV > 0: prevIntV -= 1

elif prevIntC > 0:

prevIntC -= 1

try: prevIntV = self.getNumVerses( prevBBB, prevIntC )

except KeyError:

if prevIntC != 0:

logging.error( \_("BaxBoxAddon.getBeforeAndAfterBaxData1 failed at {} {}").format( prevBBB, prevIntC ) )

failed = True

else:

try: prevBBB = self.BaxOrganisationalSystem.getPreviousBookCode( BBB )

except KeyError: prevBBB = None

if prevBBB is None: failed = True

else:

prevIntC = self.getNumChapters( prevBBB )

prevIntV = self.getNumVerses( prevBBB, prevIntC )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " Went back to previous book", prevBBB, prevIntC, prevIntV, "from", BBB, C, V )

if prevIntC is None or prevIntV is None:

logging.error( \_("BaxBoxAddon.getBeforeAndAfterBaxData2 failed at {} {}:{}").format( prevBBB, prevIntC, prevIntV ) )

break

if not failed and prevIntV is not None:

assert prevBBB and isinstance(prevBBB, str)

previousVerseKey = SimpleVerseKey( prevBBB, prevIntC, prevIntV )

previousVerseData = self.getCachedVerseData( previousVerseKey )

if previousVerseData: previousVersesData.insert( 0, (previousVerseKey,previousVerseData,) )

nextBBB, nextIntC, nextIntV = BBB, intC, intV

nextVersesData = []

for n in range( 0, self.parentApp.viewVersesAfter ):

try: numVerses = self.getNumVerses( nextBBB, nextIntC )

except KeyError: numVerses = None

nextIntV += 1

if numVerses is None or nextIntV > numVerses:

nextIntV = 1

nextIntC += 1

nextVerseKey = SimpleVerseKey( nextBBB, nextIntC, nextIntV )

nextVerseData = self.getCachedVerseData( nextVerseKey )

if nextVerseData: nextVersesData.append( (nextVerseKey,nextVerseData,) )

verseData = self.getCachedVerseData( newVerseKey )

return verseData, previousVersesData, nextVersesData

def doBaxFind( self, event=None ):

"""

获取搜索参数

"""

from BaxlatorDialogs import GetBaxFindTextDialog

self.parentApp.logUsage( ProgName, debuggingThisModule, 'BaxBoxAddon doBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.doBaxFind( {} )".format( event ) )

try: haveInternalBax = self.internalBax is not None

except AttributeError: haveInternalBax = False

if not haveInternalBax:

logging.critical( \_("No Bax to search") )

return

print( "intBib", self.internalBax )

self.BaxFindOptionsDict['currentBCV'] = self.currentVerseKey.getBCV()

gBSTD = GetBaxFindTextDialog( self, self.internalBax, self.BaxFindOptionsDict, title=\_('Find in Bax') )

if BaxOrgSysGlobals.debugFlag: print( "gBSTDResult", repr(gBSTD.result) )

if gBSTD.result:

if BaxOrgSysGlobals.debugFlag: assert isinstance( gBSTD.result, dict )

self.BaxFindOptionsDict = gBSTD.result

self.doActualBaxFind()

self.parentApp.setReadyStatus()

return tkBREAK

def doActualBaxFind( self, extendTo=None ):

from ChildWindows import FindResultWindow

self.parentApp.logUsage( ProgName, debuggingThisModule, 'BaxBoxAddon doActualBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.doActualBaxFind( {} )".format( extendTo ) )

self.parentApp.setWaitStatus( \_("Searching…") )

self.textBox.update()

self.textBox.focus()

self.lastfind = key

self.parentApp.logUsage( ProgName, debuggingThisModule, ' doActualBaxFind {}'.format( self.BaxFindOptionsDict ) )

print( "bookList", repr(self.BaxFindOptionsDict['bookList']) )

bookCode = None

if isinstance( self.BaxFindOptionsDict['bookList'], str ) \

and self.BaxFindOptionsDict['bookList'] != 'ALL':

bookCode = self.BaxFindOptionsDict['bookList']

self.\_prepareInternalBax( bookCode, self.BaxFindOptionsDict['givenBax'] )

self.BaxFindOptionsDict, resultSummaryDict, findResultList = self.BaxFindOptionsDict['givenBax'].findText( self.BaxFindOptionsDict )

print( "Got findResultList", findResultList )

if len(findResultList) == 0:

errorBeep()

key = self.BaxFindOptionsDict['findText']

showError( self, APP\_NAME, \_("String {!r} not found").format( key if len(key)<20 else (key[:18]+'…') ) )

else:

try: replaceFunction = self.doBaxReplace

except AttributeError: replaceFunction = None

findResultWindow = FindResultWindow( self, self.BaxFindOptionsDict, resultSummaryDict, findResultList,

findFunction=self.doBaxFind, refindFunction=self.doActualBaxFind,

replaceFunction=replaceFunction, extendTo=extendTo )

self.parentApp.childWindows.append( findResultWindow )

self.parentApp.setReadyStatus()

def \_prepareInternalBax( self, bookCode=None, givenBax=None ):

logging.debug( "BaxBoxAddon.\_prepareInternalBax()" )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBoxAddon.\_prepareInternalBax()" )

if givenBax is None: givenBax = self.internalBax

if self.modified(): self.doSave()

if givenBax is not None:

self.parentApp.setWaitStatus( \_("Preparing internal Bax…") )

if bookCode is None:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax…") )

givenBax.load()

else:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax book…") )

givenBax.loadBook( bookCode )

class BaxBox( ChildBox ):

def \_\_init\_\_( self, parentApp ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.\_\_init\_\_( {} )".format( parentApp ) )

assert parentApp

self.parentApp = parentApp

ChildBox.\_\_init\_\_( self, parentApp )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.\_\_init\_\_ finished." )

def createStandardBoxKeyboardBindings( self, reset=False ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.createStandardBoxKeyboardBindings( {} )".format( reset ) )

if reset:

self.myKeyboardBindingsList = []

for name,commandFunction in ( ('SelectAll',self.doSelectAll),

('Find',self.doBaxFind),

('Help',self.doHelp), ('Info',self.doShowInfo), ('About',self.doAbout),

('ShowMain',self.doShowMainWindow),

('Close',self.doClose), ):

self.\_createStandardBoxKeyboardBinding( name, commandFunction )

def createContextMenu( self ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.createContextMenu()" )

self.textBox.contextMenu = tk.Menu( self, tearoff=0 )

self.textBox.contextMenu.add\_command( label=\_('Copy'), underline=0, command=self.doCopy, accelerator=self.parentApp.keyBindingDict[\_('Copy')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Select all'), underline=7, command=self.doSelectAll, accelerator=self.parentApp.keyBindingDict[\_('SelectAll')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Bax Find…'), underline=6, command=self.doBaxFind, accelerator=self.parentApp.keyBindingDict[\_('Find')][0] )

self.textBox.contextMenu.add\_separator()

self.textBox.contextMenu.add\_command( label=\_('Find in window…'), underline=8, command=self.doBoxFind )

self.contextMenu.add\_separator()

self.contextMenu.add\_command( label=\_('Close window'), underline=1, command=self.doClose, accelerator=self.parentApp.keyBindingDict[\_('Close')][0] )

self.textBox.bind( '<Button-3>', self.showContextMenu )

self.pack()

self.BaxFindOptionsDict, self.BaxReplaceOptionsDict = {}, {}

def showContextMenu( self, event ):

self.textBox.contextMenu.tk\_popup( event.x\_root, event.y\_root )

def displayAppendVerse( self, firstFlag, verseKey, verseContextData, lastFlag=True, currentVerseFlag=False, substituteTrailingSpaces=False, substituteMultipleSpaces=False ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "displayAppendVerse( {}, {}, {}, {}, {}, {}, {} )".format( firstFlag, verseKey, verseContextData, lastFlag, currentVerseFlag, substituteTrailingSpaces, substituteMultipleSpaces ) )

assert isinstance( firstFlag, bool )

assert isinstance( verseKey, SimpleVerseKey )

if verseContextData:

assert isinstance( verseContextData, tuple ) and len(verseContextData)==2 or isinstance( verseContextData, str )

assert isinstance( lastFlag, bool )

assert isinstance( currentVerseFlag, bool )

def insertAtEnd( ieText, ieTags ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "insertAtEnd( {!r}, {} )".format( ieText, ieTags ) )

assert isinstance( ieText, str )

assert isinstance( ieTags, (str,tuple) )

assert TRAILING\_SPACE\_SUBSTITUTE not in ieText

assert MULTIPLE\_SPACE\_SUBSTITUTE not in ieText

if substituteMultipleSpaces:

ieText = ieText.replace( ' ', DOUBLE\_SPACE\_SUBSTITUTE )

ieText = ieText.replace( CLEANUP\_LAST\_MULTIPLE\_SPACE, DOUBLE\_SPACE\_SUBSTITUTE )

if substituteTrailingSpaces:

ieText = ieText.replace( TRAILING\_SPACE\_LINE, TRAILING\_SPACE\_LINE\_SUBSTITUTE )

self.textBox.insert( tk.END, ieText, ieTags )

try: cVM, fVM = self.\_contextViewMode, self.\_formatViewMode

except AttributeError:

cVM, fVM = self.parentWindow.\_contextViewMode, self.parentWindow.\_formatViewMode

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "displayAppendVerse2( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.displayAppendVerse( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

try: print( "BaxBox.displayAppendVerse( {}, {}, {}, {} )".format( firstFlag, verseKey, verseContextData, currentVerseFlag ) )

except UnicodeEncodeError: print( "BaxBox.displayAppendVerse", firstFlag, verseKey, currentVerseFlag )

BBB, C, V = verseKey.getBCV()

C, V = int(C), int(V)

C1 = C2 = int(C); V1 = V2 = int(V)

if V1 > 0: V1 -= 1

elif C1 > 0:

C1 -= 1

V1 = self.getNumVerses( BBB, C1 )

if V2 < self.getNumVerses( BBB, C2 ): V2 += 1

elif C2 < self.getNumChapters( BBB):

C2 += 1

V2 = 0

previousMarkName = 'C{}V{}'.format( C1, V1 )

currentMarkName = 'C{}V{}'.format( C, V )

nextMarkName = 'C{}V{}'.format( C2, V2 )

print( "Marks", previousMarkName, currentMarkName, nextMarkName )

lastCharWasSpace = haveTextFlag = not firstFlag

if verseContextData is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "displayAppendVerse has no data for", verseKey )

verseDataList = context = None

elif isinstance( verseContextData, tuple ):

assert len(verseContextData) == 2

verseDataList, context = verseContextData

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " VerseDataList: {}".format( verseDataList ) )

print( " Context: {}".format( context ) )

elif isinstance( verseContextData, str ):

verseDataList, context = verseContextData.split( '\n' ), None

elif BaxOrgSysGlobals.debugFlag: halt

if firstFlag:

if context:

print( "context", context )

print( " Setting context mark to {}".format( previousMarkName ) )

self.textBox.mark\_set( previousMarkName, tk.INSERT )

self.textBox.mark\_gravity( previousMarkName, tk.LEFT )

insertAtEnd( ' '+\_("Prior context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in context:

print( " someMarker", someMarker )

if someMarker != 'chapters':

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

haveTextFlag = True

if verseDataList and fVM == 'Formatted':

firstEntry = verseDataList[0]

if isinstance( firstEntry, InternalBaxEntry ): marker = firstEntry.getMarker()

elif isinstance( firstEntry, tuple ): marker = firstEntry[0]

else: marker = None

if marker in BaxOrgSysGlobals.USFMParagraphMarkers:

insertAtEnd( ' '+\_("Current context")+': ', 'contextHeader' )

insertAtEnd( marker+' ', 'context' )

Display all line markers in this segment

markerList = []

for verseData in verseDataList:

if isinstance( verseData, InternalBaxEntry ): marker = verseData.getMarker()

elif isinstance( verseData, tuple ): marker = verseData[0]

else: marker = None

if marker and not marker.startswith('¬') \

and not marker.endswith('~') and not marker.endswith('#'):

markerList.append( marker )

if markerList:

insertAtEnd( ' '+\_("Displayed markers")+': ', 'markersHeader' )

insertAtEnd( str(markerList)[1:-1], 'markers' )

print( " Setting mark to {}".format( currentMarkName ) )

self.textBox.mark\_set( currentMarkName, tk.INSERT )

self.textBox.mark\_gravity( currentMarkName, tk.LEFT )

if verseDataList is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "BaxBox.displayAppendVerse has no data for", self.moduleID, verseKey )

self.textBox.insert( tk.END, '--' )

else:

hadVerseText = False

try: cVM = self.\_contextViewMode

except AttributeError: cVM = self.parentWindow.\_contextViewMode

lastParagraphMarker = context[-1] if context and context[-1] in BaxOrgSysGlobals.USFMParagraphMarkers \

else 'v~'

endMarkers = []

for verseDataEntry in verseDataList:

if isinstance( verseDataEntry, InternalBaxEntry ):

marker, cleanText = verseDataEntry.getMarker(), verseDataEntry.getCleanText()

elif isinstance( verseDataEntry, tuple ):

marker, cleanText = verseDataEntry[0], verseDataEntry[3]

elif isinstance( verseDataEntry, str ):

if verseDataEntry=='': continue

verseDataEntry += '\n'

if verseDataEntry[0]=='\\':

marker = ''

for char in verseDataEntry[1:]:

if char!='¬' and not char.isalnum(): break

marker += char

cleanText = verseDataEntry[len(marker)+1:].lstrip()

else:

marker, cleanText = None, verseDataEntry

elif BaxOrgSysGlobals.debugFlag: halt

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " displayAppendVerse", lastParagraphMarker, haveTextFlag, marker, repr(cleanText) )

if fVM == 'Unformatted':

if marker and marker[0]=='¬': pass

elif marker in ('intro','chapters','list',): pass

else:

if isinstance( verseDataEntry, str ):

print( "marker={!r}, verseDataEntry={!r}".format( marker, verseDataEntry ) )

insertAtEnd( verseDataEntry, marker )

else:

if hadVerseText and marker in ( 's', 's1', 's2', 's3' ):

print( " Setting s mark to {}".format( nextMarkName ) )

self.textBox.mark\_set( nextMarkName, tk.INSERT )

self.textBox.mark\_gravity( nextMarkName, tk.LEFT )

print( " Inserting ({}): {!r}".format( marker, verseDataEntry ) )

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

if marker is None:

insertAtEnd( cleanText, '###' )

else: insertAtEnd( '\\{} {}'.format( marker, cleanText ), marker+'#' )

haveTextFlag = True

elif fVM == 'Formatted':

if marker.startswith( '¬' ):

if marker != '¬v': endMarkers.append( marker )

else: endMarkers = []

if marker.startswith( '¬' ):

pass

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker == 'id':

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ide','rem',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('h','toc1','toc2','toc3','cl¤',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('intro','chapters','list',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('mt1','mt2','mt3','mt4', 'imt1','imt2','imt3','imt4', 'iot','io1','io2','io3','io4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ip','ipi','im','imi','ipq','imq','ipr', 'iq1','iq2','iq3','iq4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('s1','s2','s3','s4', 'is1','is2','is3','is4', 'ms1','ms2','ms3','ms4', 'cl',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('d','sp',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('r','mr','sr',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in BaxOrgSysGlobals.USFMParagraphMarkers:

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

lastParagraphMarker = marker

haveTextFlag = True

elif marker in ('b','ib'):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

elif marker in ('m','im'):

self.textBox.insert ( tk.END, '\n' if haveTextFlag else ' ', marker )

if cleanText:

insertAtEnd( cleanText, '\*'+marker if currentVerseFlag else marker )

lastCharWasSpace = False

haveTextFlag = True

elif marker == 'p#' and self.BaxBoxType=='DBPBaxResourceBox':

pass

elif marker == 'c':

if not firstFlag: haveC = cleanText

else: print( " Ignore C={}".format( cleanText ) )

pass

elif marker == 'c#':

if cleanText != verseKey.getBBB():

if not lastCharWasSpace: insertAtEnd( ' ', 'v-' )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

lastCharWasSpace = False

elif marker == 'v':

if cleanText != '1':

if haveTextFlag:

insertAtEnd( ' ', (lastParagraphMarker,'v-',) if lastParagraphMarker else ('v-',) )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

insertAtEnd( '\u2009', (lastParagraphMarker,'v+',) if lastParagraphMarker else ('v+',) )

lastCharWasSpace = haveTextFlag = True

elif marker in ('v~','p~'):

insertAtEnd( cleanText, '\*'+lastParagraphMarker if currentVerseFlag else lastParagraphMarker )

haveTextFlag = True

else:

if BaxOrgSysGlobals.debugFlag:

logging.critical( \_("BaxBox.displayAppendVerse (formatted): Unknown marker {!r} {!r} from {}").format( marker, cleanText, verseDataList ) )

else:

logging.critical( \_("BaxBox.displayAppendVerse (formatted): Unknown marker {!r} {!r}").format( marker, cleanText ) )

else:

logging.critical( \_("BaxBox.displayAppendVerse: Unknown {!r} format view mode").format( fVM ) )

if BaxOrgSysGlobals.debugFlag: halt

if lastFlag and cVM=='ByVerse' and endMarkers:

print( "endMarkers", endMarkers )

insertAtEnd( ' '+ \_("End context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in endMarkers:

print( " someMarker", someMarker )

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

def getBeforeAndAfterBaxData( self, newVerseKey ):

if BaxOrgSysGlobals.debugFlag:

print( "BaxBox.getBeforeAndAfterBaxData( {} )".format( newVerseKey ) )

assert isinstance( newVerseKey, SimpleVerseKey )

BBB, C, V = newVerseKey.getBCV()

intC, intV = newVerseKey.getChapterNumberInt(), newVerseKey.getVerseNumberInt()

prevBBB, prevIntC, prevIntV = BBB, intC, intV

previousVersesData = []

for n in range( -self.parentApp.viewVersesBefore, 0 ):

failed = False

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " getBeforeAndAfterBaxData here with", n, prevIntC, prevIntV )

if prevIntV > 0: prevIntV -= 1

elif prevIntC > 0:

prevIntC -= 1

try: prevIntV = self.getNumVerses( prevBBB, prevIntC )

except KeyError:

if prevIntC != 0:

logging.error( \_("BaxBox.getBeforeAndAfterBaxData1 failed at {} {}").format( prevBBB, prevIntC ) )

failed = True

if not failed:

if BaxOrgSysGlobals.debugFlag: print( " Went back to previous chapter", prevIntC, prevIntV, "from", BBB, C, V )

else:

try: prevBBB = self.BaxOrganisationalSystem.getPreviousBookCode( BBB )

except KeyError: prevBBB = None

if prevBBB is None: failed = True

else:

prevIntC = self.getNumChapters( prevBBB )

prevIntV = self.getNumVerses( prevBBB, prevIntC )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " Went back to previous book", prevBBB, prevIntC, prevIntV, "from", BBB, C, V )

if prevIntC is None or prevIntV is None:

logging.error( \_("BaxBox.getBeforeAndAfterBaxData2 failed at {} {}:{}").format( prevBBB, prevIntC, prevIntV ) )

failed = True

break

if not failed and prevIntV is not None:

print( "getBeforeAndAfterBaxData XXX", repr(prevBBB), repr(prevIntC), repr(prevIntV) )

assert prevBBB and isinstance(prevBBB, str)

previousVerseKey = SimpleVerseKey( prevBBB, prevIntC, prevIntV )

previousVerseData = self.getCachedVerseData( previousVerseKey )

if previousVerseData: previousVersesData.insert( 0, (previousVerseKey,previousVerseData,) )

nextBBB, nextIntC, nextIntV = BBB, intC, intV

nextVersesData = []

for n in range( 0, self.parentApp.viewVersesAfter ):

try: numVerses = self.getNumVerses( nextBBB, nextIntC )

except KeyError: numVerses = None

nextIntV += 1

if numVerses is None or nextIntV > numVerses:

nextIntV = 1

nextIntC += 1

nextVerseKey = SimpleVerseKey( nextBBB, nextIntC, nextIntV )

nextVerseData = self.getCachedVerseData( nextVerseKey )

if nextVerseData: nextVersesData.append( (nextVerseKey,nextVerseData,) )

verseData = self.getCachedVerseData( newVerseKey )

return verseData, previousVersesData, nextVersesData

def doBaxFind( self, event=None ):

from BaxlatorDialogs import GetBaxFindTextDialog

self.parentApp.logUsage( ProgName, debuggingThisModule, 'BaxBox doBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.doBaxFind( {} )".format( event ) )

try: haveInternalBax = self.internalBax is not None

except AttributeError: haveInternalBax = False

if not haveInternalBax:

logging.critical( \_("No Bax to search") )

return

print( "intBib", self.internalBax )

self.BaxFindOptionsDict['currentBCV'] = self.currentVerseKey.getBCV()

gBSTD = GetBaxFindTextDialog( self, self.internalBax, self.BaxFindOptionsDict, title=\_('Find in Bax') )

if BaxOrgSysGlobals.debugFlag: print( "gBSTDResult", repr(gBSTD.result) )

if gBSTD.result:

if BaxOrgSysGlobals.debugFlag: assert isinstance( gBSTD.result, dict )

self.BaxFindOptionsDict = gBSTD.result

self.doActualBaxFind()

self.parentApp.setReadyStatus()

return tkBREAK

def doActualBaxFind( self, extendTo=None ):

from ChildWindows import FindResultWindow

self.parentApp.logUsage( ProgName, debuggingThisModule, 'BaxBox doActualBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.doActualBaxFind( {} )".format( extendTo ) )

self.parentApp.setWaitStatus( \_("Searching…") )

self.textBox.update()

self.textBox.focus()

self.lastfind = key

self.parentApp.logUsage( ProgName, debuggingThisModule, ' doActualBaxFind {}'.format( self.BaxFindOptionsDict ) )

print( "bookList", repr(self.BaxFindOptionsDict['bookList']) )

bookCode = None

if isinstance( self.BaxFindOptionsDict['bookList'], str ) \

and self.BaxFindOptionsDict['bookList'] != 'ALL':

bookCode = self.BaxFindOptionsDict['bookList']

self.\_prepareInternalBax( bookCode, self.BaxFindOptionsDict['givenBax'] )

self.BaxFindOptionsDict, resultSummaryDict, findResultList = self.BaxFindOptionsDict['givenBax'].findText( self.BaxFindOptionsDict )

print( "Got findResults", findResults )

if len(findResultList) == 0:

errorBeep()

key = self.BaxFindOptionsDict['findText']

showError( self, APP\_NAME, \_("String {!r} not found").format( key if len(key)<20 else (key[:18]+'…') ) )

else:

try: replaceFunction = self.doBaxReplace

except AttributeError: replaceFunction = None

findResultWindow = FindResultWindow( self, self.BaxFindOptionsDict, resultSummaryDict, findResultList,

findFunction=self.doBaxFind, refindFunction=self.doActualBaxFind,

replaceFunction=replaceFunction, extendTo=extendTo )

self.parentApp.childWindows.append( findResultWindow )

self.parentApp.setReadyStatus()

def \_prepareInternalBax( self, bookCode=None, givenBax=None ):

logging.debug( "BaxBox.\_prepareInternalBax()" )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "BaxBox.\_prepareInternalBax()" )

if givenBax is None: givenBax = self.internalBax

if self.modified(): self.doSave()

if givenBax is not None:

self.parentApp.setWaitStatus( \_("Preparing internal Bax…") )

if bookCode is None:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax…") )

givenBax.load()

else:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax book…") )

givenBax.loadBook( bookCode )

class HebrewInterlinearBaxBoxAddon( BaxBoxAddon ):

def \_\_init\_\_( self, parentWindow, numInterlinearLines ):

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.\_\_init\_\_( {}, nIL={} )".format( parentWindow, numInterlinearLines ) )

assert parentWindow

assert 0 < numInterlinearLines <= 5

self.numInterlinearLines = numInterlinearLines

BaxBoxAddon.\_\_init\_\_( self, parentWindow, 'HebrewInterlinearBaxBoxAddon' )

self.tabStopCm = 3.0

self.textBox.config( tabs='3.0c' )

self.pixelsPerCm = self.parentWindow.parentApp.rootWindow.winfo\_fpixels( '1c' )

self.tabStopPixels = self.tabStopCm \* self.pixelsPerCm

self.entryStylesNormal = ( 'HebWord', 'HebStrong', 'HebMorph', 'HebGenericGloss', 'HebSpecificGloss' )

self.entryStylesSelected = ( 'HebWordSelected', 'HebStrongSelected', 'HebMorphSelected', 'HebGenericGlossSelected', 'HebSpecificGlossSelected' )

self.fontsNormal, self.fontsSelected = [], []

for entryStyleNormal,entryStyleSelected in zip( self.entryStylesNormal, self.entryStylesSelected ):

fontNormal = tkFont.Font( \*\*self.parentWindow.parentApp.stylesheet.getTKStyleDict( entryStyleNormal ) )

fontSelected = tkFont.Font( \*\*self.parentWindow.parentApp.stylesheet.getTKStyleDict( entryStyleSelected ) )

self.fontsNormal.append( fontNormal )

self.fontsSelected.append( fontSelected )

tabWidthNormal = fontNormal.measure( ' '\*8 )

tabWidthSelected = fontSelected.measure( ' '\*8 )

print( "tabWidths", tabWidthNormal, tabWidthSelected )

tabWidthsNormal.append( tabWidthNormal )

tabWidthsSelected.append( tabWidthSelected )

self.requestMissingGlosses, self.glossWindowGeometry = True, None

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.\_\_init\_\_ finished." )

def displayAppendVerse( self, firstFlag, verseKey, verseContextData, lastFlag=True, currentVerseFlag=False, currentWordNumber=1, command=None, substituteTrailingSpaces=False, substituteMultipleSpaces=False ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.displayAppendVerse( fF={}, {}, {}, lF={}, cVF={}, cWN={}, c={}, sTS={}, sMS={} )" \

.format( firstFlag, verseKey, verseContextData, lastFlag, currentVerseFlag, currentWordNumber, command, substituteTrailingSpaces, substituteMultipleSpaces ) )

print( " {}".format( verseContextData[0] ) )

assert isinstance( firstFlag, bool )

assert isinstance( verseKey, SimpleVerseKey )

if verseContextData:

assert isinstance( verseContextData, tuple ) and len(verseContextData)==2 or isinstance( verseContextData, str )

assert isinstance( lastFlag, bool )

assert isinstance( currentVerseFlag, bool )

self.lastDAVargs = firstFlag, verseKey, verseContextData, lastFlag, currentVerseFlag, currentWordNumber, None, substituteTrailingSpaces, substituteMultipleSpaces

self.update()

boxWidth = self.textBox.winfo\_width()

print( "boxWidth", boxWidth )

self.bundlesPerLine = int( boxWidth / (self.tabStopCm \* self.pixelsPerCm) ) + 1

print( "bundlesPerLine", self.bundlesPerLine )

def insertAtEnd( ieText, ieTags ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "insertAtEnd( {!r}, {} )".format( ieText, ieTags ) )

assert isinstance( ieText, str )

assert ieTags is None or isinstance( ieTags, (str,tuple) )

assert TRAILING\_SPACE\_SUBSTITUTE not in ieText

assert MULTIPLE\_SPACE\_SUBSTITUTE not in ieText

if substituteMultipleSpaces:

ieText = ieText.replace( ' ', DOUBLE\_SPACE\_SUBSTITUTE )

ieText = ieText.replace( CLEANUP\_LAST\_MULTIPLE\_SPACE, DOUBLE\_SPACE\_SUBSTITUTE )

if substituteTrailingSpaces:

ieText = ieText.replace( TRAILING\_SPACE\_LINE, TRAILING\_SPACE\_LINE\_SUBSTITUTE )

self.textBox.insert( tk.END, ieText, ieTags )

def insertAtEndLine( ieLineNumber, ieText, ieTags ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "insertAtEndLine( {}, {!r}, {} )".format( ieLineNumber, ieText, ieTags ) )

assert isinstance( ieLineNumber, int )

assert isinstance( ieText, str )

assert ieTags is None or isinstance( ieTags, (str,tuple) )

assert TRAILING\_SPACE\_SUBSTITUTE not in ieText

assert MULTIPLE\_SPACE\_SUBSTITUTE not in ieText

if substituteMultipleSpaces:

ieText = ieText.replace( ' ', DOUBLE\_SPACE\_SUBSTITUTE )

ieText = ieText.replace( CLEANUP\_LAST\_MULTIPLE\_SPACE, DOUBLE\_SPACE\_SUBSTITUTE )

if substituteTrailingSpaces:

ieText = ieText.replace( TRAILING\_SPACE\_LINE, TRAILING\_SPACE\_LINE\_SUBSTITUTE )

self.textBox.mark\_set( tk.INSERT, '{}.0 lineend'.format( ieLineNumber ) )

self.textBox.insert( tk.INSERT, ieText, ieTags )

def appendVerseText( verseDataEntry, currentVerseKey, currentVerseFlag, currentWordNumber, command ):

from BaxlatorDialogs import GetHebrewGlossWordDialog, GetHebrewGlossWordsDialog

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "displayAppendVerse.appendVerseText( {}, {}, cVF={}, cWN={}, c={} )".format( verseDataEntry, currentVerseKey, currentVerseFlag, currentWordNumber, command ) )

verseDictList = self.internalBax.getVerseDictList( verseDataEntry, currentVerseKey )

print( verseKey.getShortText(), "verseDictList", verseDictList )

self.textBox.insert( tk.END, '\n'\*self.numInterlinearLines )

insertAtEnd( '\n'\*self.numInterlinearLines, None )

haveTextFlag = False

savedLineNumber = self.lineNumber

requestMissingGlossesNow = needToRequestMissingGlosses = needToUpdate = False

for passNumber in range( 1, 3 ):

print( "HebrewInterlinearBaxBoxAddon.appendVerseText: pass #{} {} {}".format( passNumber, requestMissingGlossesNow, needToRequestMissingGlosses ) )

self.lineNumber = savedLineNumber

self.acrossIndex = 0

j = 0

while j < len(verseDictList):

j += 1

verseDict = verseDictList[j-1]

print( "pn={}, j={}, c={}, verseDict={}".format( passNumber, j, command, verseDict ) )

if bundlesAcross >= self.bundlesPerLine:

print( "Start new bundle line" )

self.textBox.insert( tk.END, '\n'\*(self.numInterlinearLines+1) )

insertAtEnd( '\n'\*(self.numInterlinearLines+1) )

self.lineNumber += self.numInterlinearLines + 1

bundlesAcross = 0

haveTextFlag = False

word = verseDict['word']

fullRefTuple = currentVerseKey.getBCV() + (str(j),)

refText = '{} {}:{}.{}'.format( \*fullRefTuple )

import Hebrew

h = Hebrew.Hebrew( word )

print( '{!r} is '.format( word ), end=None )

h.printUnicodeData( word )

try: strongsNumber = verseDict['strong']

except KeyError: strongsNumber = ''

try: morphology = verseDict['morph']

except KeyError: morphology = ''

if self.numInterlinearLines == 3:

bundle = word, strongsNumber, morphology, self.internalBax.expandMorphologyAbbreviations( morphology )

elif self.numInterlinearLines == 4:

assert self.internalBax.glossingDict

normalizedWord = self.internalBax.removeCantillationMarks( word, removeMetegOrSiluq=True ) \

.replace( ORIGINAL\_MORPHEME\_BREAK\_CHAR, OUR\_MORPHEME\_BREAK\_CHAR )

if normalizedWord != word:

print( ' ({}) {!r} normalized to ({}) {!r}'.format( len(word), word, len(normalizedWord), normalizedWord ) )

print( '{!r} is '.format( normalizedWord ), end=None )

h.printUnicodeData( normalizedWord )

genericGloss,genericReferencesList,specificReferencesDict = self.internalBax.glossingDict[normalizedWord] \

if normalizedWord in self.internalBax.glossingDict else ('',[],{})

if passNumber>1 and ( command in ('L','R') or (command=='E' and j==currentWordNumber) ):

command = None

tempBundle = refText, normalizedWord, strongsNumber, morphology, self.internalBax.expandMorphologyAbbreviations( morphology )

self.parentWindow.setStatus( self.internalBax.expandMorphologyAbbreviations( morphology ) )

ghgwd = GetHebrewGlossWordDialog( self, \_("Edit generic gloss"), tempBundle, genericGloss, geometry=self.glossWindowGeometry )

print( "ghgwdResultA1", ghgwd.result )

if ghgwd.result is None: # 取消

self.requestMissingGlosses = False

elif ghgwd.result == 'S': # 跳过

needToRequestMissingGlosses = False

elif ghgwd.result in ('L','R','LL','RR'): # 向左/右偏

command = ghgwd.result

elif isinstance( ghgwd.result, dict ):

print( "result1", ghgwd.result )

assert ghgwd.result['word']

genericGloss = ghgwd.result['word']

self.internalBax.setNewGenericGloss( normalizedWord, genericGloss, fullRefTuple )

self.glossWindowGeometry = ghgwd.result['geometry'] # 保持窗口位置及大小

try: command = ghgwd.result['command'] # 左右

except KeyError: command = None

needToRequestMissingGlosses = False

needToUpdate = True

else: halt # 程序错误

elif not genericGloss and BaxOrgSysGlobals.verbosityLevel > 0:

print( "No generic gloss found for ({}) {}{}".format( len(word), word, \

' to ({}) {}'.format( len(normalizedWord), normalizedWord ) if normalizedWord!=word else '' ) )

if self.requestMissingGlosses and requestMissingGlossesNow and not self.parentApp.starting:

tempBundle = refText, normalizedWord, strongsNumber, morphology, self.internalBax.expandMorphologyAbbreviations( morphology )

self.parentWindow.setStatus( self.internalBax.expandMorphologyAbbreviations( morphology ) )

ghgwd = GetHebrewGlossWordDialog( self, \_("Enter new generic gloss"), tempBundle, geometry=self.glossWindowGeometry )

print( "ghgwdResultA2", ghgwd.result )

if ghgwd.result is None: # 取消

self.requestMissingGlosses = False

elif ghgwd.result == 'S': # 跳过

needToRequestMissingGlosses = False

elif ghgwd.result in ('L','R','LL','RR'): # 左右留空

command = ghgwd.result

elif isinstance( ghgwd.result, dict ):

print( "result2", ghgwd.result )

assert ghgwd.result['word']

genericGloss = ghgwd.result['word']

self.internalBax.setNewGenericGloss( normalizedWord, genericGloss, fullRefTuple )

self.glossWindowGeometry = ghgwd.result['geometry']

try: command = ghgwd.result['command']

except KeyError: command = None

needToRequestMissingGlosses = False

needToUpdate = True

else: halt # 程序错误

else: needToRequestMissingGlosses = True

bundle = word, strongsNumber, morphology, genericGloss

elif self.numInterlinearLines == 5:

assert self.internalBax.glossingDict

normalizedWord = self.internalBax.removeCantillationMarks( word, removeMetegOrSiluq=True ) \

.replace( ORIGINAL\_MORPHEME\_BREAK\_CHAR, OUR\_MORPHEME\_BREAK\_CHAR )

if normalizedWord != word:

print( ' ({}) {!r} normalized to ({}) {!r}'.format( len(word), word, len(normalizedWord), normalizedWord ) )

print( '{!r} is '.format( normalizedWord ), end=None )

h.printUnicodeData( normalizedWord )

genericGloss,genericReferencesList,specificReferencesDict = self.internalBax.glossingDict[normalizedWord] \

if normalizedWord in self.internalBax.glossingDict else ('',[],{})

try: specificGloss = specificReferencesDict[fullRefTuple]

except KeyError: specificGloss = ''

if passNumber>1 and ( command in ('L','R') or (command=='E' and j==currentWordNumber) ):

command = None

tempBundle = refText, normalizedWord, strongsNumber, morphology, self.internalBax.expandMorphologyAbbreviations( morphology )

self.parentWindow.setStatus( self.internalBax.expandMorphologyAbbreviations( morphology ) )

ghgwd = GetHebrewGlossWordsDialog( self, \_("Edit generic/specific glosses"), tempBundle, genericGloss, specificGloss, geometry=self.glossWindowGeometry )

print( "ghgwdResultB1", ghgwd.result )

if ghgwd.result is None:

self.requestMissingGlosses = False

elif ghgwd.result == 'S':

needToRequestMissingGlosses = False

elif ghgwd.result in ('L','R','LL','RR'):

command = ghgwd.result

elif isinstance( ghgwd.result, dict ):

print( "result3", ghgwd.result )

assert ghgwd.result['word1']

genericGloss = ghgwd.result['word1']

specificGloss = ghgwd.result['word2'] if 'word2' in ghgwd.result else None

self.internalBax.setNewGenericGloss( normalizedWord, genericGloss, fullRefTuple )

if specificGloss:

self.internalBax.setNewSpecificGloss( normalizedWord, specificGloss, fullRefTuple )

self.glossWindowGeometry = ghgwd.result['geometry']

try: command = ghgwd.result['command']

except KeyError: command = None

needToRequestMissingGlosses = False

needToUpdate = True

else: halt

elif not genericGloss and BaxOrgSysGlobals.verbosityLevel > 0:

print( "No generic gloss found for ({}) {}{}".format( len(word), word, \

' to ({}) {}'.format( len(normalizedWord), normalizedWord ) if normalizedWord!=word else '' ) )

if self.requestMissingGlosses and requestMissingGlossesNow and not self.parentApp.starting:

tempBundle = refText, normalizedWord, strongsNumber, morphology, self.internalBax.expandMorphologyAbbreviations( morphology )

self.parentWindow.setStatus( self.internalBax.expandMorphologyAbbreviations( morphology ) )

ghgwd = GetHebrewGlossWordsDialog( self, \_("Enter new generic/specific glosses"), tempBundle, geometry=self.glossWindowGeometry )

print( "ghgwdResultB2", ghgwd.result )

if ghgwd.result is None: # 取消

self.requestMissingGlosses = False

elif ghgwd.result == 'S': # 跳过

needToRequestMissingGlosses = False

elif ghgwd.result in ('L','R','LL','RR'):

command = ghgwd.result

elif isinstance( ghgwd.result, dict ):

print( "result4", ghgwd.result )

assert ghgwd.result['word1']

genericGloss = ghgwd.result['word1']

specificGloss = ghgwd.result['word2'] if 'word2' in ghgwd.result else None

self.internalBax.setNewGenericGloss( normalizedWord, genericGloss, fullRefTuple )

if specificGloss:

self.internalBax.setNewSpecificGloss( normalizedWord, specificGloss, fullRefTuple )

self.glossWindowGeometry = ghgwd.result['geometry']

try: command = ghgwd.result['command']

except KeyError: command = None

needToRequestMissingGlosses = False

needToUpdate = True

else: halt

else: needToRequestMissingGlosses = True

bundle = word, strongsNumber, morphology, genericGloss, specificGloss

else: halt

if passNumber == 1:

appendBundle( bundle, j, j==currentWordNumber, haveTextFlag )

haveTextFlag = True

if command == 'L':

if j>1: j = j - 2

else: command = None

elif command == 'R':

if j < len(verseDictList): pass

else: command = None

elif command == 'LL':

self.parentApp.doGotoPreviousVerse()

return False

elif command == 'RR':

self.parentApp.doGotoNextVerse()

return False

elif command == 'E': pass

else: assert command is None

bundlesAcross += 1

if self.parentApp.starting: break

if command: continue

if not self.requestMissingGlosses: break

if not needToRequestMissingGlosses: break

requestMissingGlossesNow = True

return needToUpdate

def appendBundle( textBundle, wordNumber, currentBundleFlag, haveTextFlag ):

if BaxOrgSysGlobals.debugFlag:

if debuggingThisModule:

print( "displayAppendVerse.appendBundle( {}, wN={}, cBF={}, hTF={} )".format( textBundle, wordNumber, currentBundleFlag, haveTextFlag ) )

assert isinstance( textBundle, tuple )

assert len(textBundle) == self.numInterlinearLines

if currentBundleFlag:

entryStyles, fonts = self.entryStylesSelected, self.fontsSelected

self.parentWindow.setStatus( self.internalBax.expandMorphologyAbbreviations(textBundle[2]) )

else:

entryStyles, fonts = self.entryStylesNormal, self.fontsNormal

maxWidthPixels = 0

bundleWidthsPixels = []

tabStopsUsed = []

for j,bundleEntry in enumerate( textBundle ):

print( "bundleEntry", bundleEntry )

(w,h) = (font.measure(text),font.metrics("linespace"))

bundleWidthPixels = fonts[j].measure( bundleEntry ) + 6

bundleWidthsPixels.append( bundleWidthPixels )

tabStopsUsed.append( int( bundleWidthPixels / self.tabStopPixels ) + 1 )

print( j, currentBundleFlag, bundleEntry, bundleWidthPixels )

if bundleWidthPixels > maxWidthPixels: maxWidthPixels = bundleWidthPixels

maxTabStopsUsed = int( maxWidthPixels / self.tabStopPixels ) + 1

if maxTabStopsUsed>1:

print( " Need more tabs bWP={} tSU={} mWP={} tSP={} mTSU={} bpL={}" \

.format( bundleWidthsPixels, tabStopsUsed, maxWidthPixels, self.tabStopPixels, maxTabStopsUsed, self.bundlesPerLine ) )

if self.acrossIndex+maxTabStopsUsed >= self.bundlesPerLine:

print( "Start new bundle line" )

self.textBox.insert( tk.END, '\n'\*(self.numInterlinearLines+1) )

insertAtEnd( '\n'\*(self.numInterlinearLines+1), None )

self.lineNumber += self.numInterlinearLines + 1

self.acrossIndex = 0

haveTextFlag = False

print( "About to display bundle {} at row={} col={}".format( textBundle, self.lineNumber, self.acrossIndex ) )

for j,(bundleEntry,bundleWidthPixels,thisTabStopsUsed) in enumerate( zip(textBundle,bundleWidthsPixels,tabStopsUsed) ):

print( "bundleEntry", bundleEntry, bundleWidthPixels )

print( "bundleEntry", bundleEntry )

if j==0: bundleEntry = bundleEntry[::-1]

wTag = 'W{}.{}'.format( wordNumber, j )

if numTabsRequired:

insertAtEndLine( self.lineNumber+j, '\t'\*numTabsRequired, self.entryStylesNormal[j] )

insertAtEndLine( self.lineNumber+j, bundleEntry, (entryStyles[j],wTag) )

self.textBox.tag\_bind( wTag, '<Button-1>', self.selectBundle )

self.textBox.tag\_bind( wTag, '<Double-Button-1>', self.editBundle )

numTabsRequired = 1

if maxTabStopsUsed > 1:

tabStopsUsed = int( bundleWidthPixels / self.tabStopPixels )

numTabsRequired += maxTabStopsUsed - thisTabStopsUsed

print( " Appending {} trailing tab{} to {!r}" \

.format( numTabsRequired, '' if numTabsRequired==1 else 's', bundleEntry ) )

insertAtEndLine( self.lineNumber+j, '\t'\*numTabsRequired, None )

self.acrossIndex += maxTabStopsUsed

return maxTabStopsUsed

needsRefreshing = False

while True:

self.lineNumber = 0

try: cVM, fVM = self.\_contextViewMode, self.\_formatViewMode

except AttributeError:

cVM, fVM = self.parentWindow.\_contextViewMode, self.parentWindow.\_formatViewMode

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "displayAppendVerse2( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

assert cVM == 'ByVerse'

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.displayAppendVerse( {}, {}, …, {}, {} ) for {}/{}".format( firstFlag, verseKey, lastFlag, currentVerseFlag, fVM, cVM ) )

try: print( "HebrewInterlinearBaxBoxAddon.displayAppendVerse( {}, {}, {}, {} )".format( firstFlag, verseKey, verseContextData, currentVerseFlag ) )

except UnicodeEncodeError: print( "HebrewInterlinearBaxBoxAddon.displayAppendVerse", firstFlag, verseKey, currentVerseFlag )

BBB, C, V = verseKey.getBCV()

C, V = int(C), int(V)

C1 = C2 = int(C); V1 = V2 = int(V)

if V1 > 0: V1 -= 1

elif C1 > 0:

C1 -= 1

V1 = self.getNumVerses( BBB, C1 )

if V2 < self.getNumVerses( BBB, C2 ): V2 += 1

elif C2 < self.getNumChapters( BBB):

C2 += 1

V2 = 0

previousMarkName = 'C{}V{}'.format( C1, V1 )

currentMarkName = 'C{}V{}'.format( C, V )

nextMarkName = 'C{}V{}'.format( C2, V2 )

print( "Marks", previousMarkName, currentMarkName, nextMarkName )

lastCharWasSpace = haveTextFlag = not firstFlag

if verseContextData is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "displayAppendVerse has no data for", verseKey )

verseDataList = context = None

elif isinstance( verseContextData, tuple ):

assert len(verseContextData) == 2

verseDataList, context = verseContextData

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " VerseDataList: {}".format( verseDataList ) )

print( " Context: {}".format( context ) )

elif isinstance( verseContextData, str ):

verseDataList, context = verseContextData.split( '\n' ), None

elif BaxOrgSysGlobals.debugFlag: halt

if firstFlag:

if context:

print( "context", context )

print( " Setting context mark to {}".format( previousMarkName ) )

self.textBox.mark\_set( previousMarkName, tk.INSERT )

self.textBox.mark\_gravity( previousMarkName, tk.LEFT )

insertAtEnd( ' '+\_("Prior context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in context:

print( " someMarker", someMarker )

if someMarker != 'chapters':

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

haveTextFlag = True

self.lineNumber += 1

if verseDataList and fVM == 'Formatted':

firstEntry = verseDataList[0]

if isinstance( firstEntry, InternalBaxEntry ): marker = firstEntry.getMarker()

elif isinstance( firstEntry, tuple ): marker = firstEntry[0]

else: marker = None

if marker in BaxOrgSysGlobals.USFMParagraphMarkers:

insertAtEnd( ' '+\_("Current context")+': ', 'contextHeader' )

insertAtEnd( marker+' ', 'context' )

Display all line markers in this segment

markerList = []

for verseData in verseDataList:

if isinstance( verseData, InternalBaxEntry ): marker = verseData.getMarker()

elif isinstance( verseData, tuple ): marker = verseData[0]

else: marker = None

if marker and not marker.startswith('¬') \

and not marker.endswith('~') and not marker.endswith('#'):

markerList.append( marker )

if markerList:

insertAtEnd( ' '+\_("Displayed markers")+': ', 'markersHeader' )

insertAtEnd( str(markerList)[1:-1], 'markers' )

self.textBox.insert ( tk.END, ' ' )

print( " Setting mark to {}".format( currentMarkName ) )

self.textBox.mark\_set( currentMarkName, tk.INSERT )

self.textBox.mark\_gravity( currentMarkName, tk.LEFT )

if verseDataList is None:

if BaxOrgSysGlobals.debugFlag and debuggingThisModule and C!=0 and V!=0:

print( " ", "HebrewInterlinearBaxBoxAddon.displayAppendVerse has no data for", self.moduleID, verseKey )

self.textBox.insert( tk.END, '--' )

else:

hadVerseText = False

try: cVM = self.\_contextViewMode

except AttributeError: cVM = self.parentWindow.\_contextViewMode

lastParagraphMarker = context[-1] if context and context[-1] in BaxOrgSysGlobals.USFMParagraphMarkers \

else 'v~'

endMarkers = []

for verseDataEntry in verseDataList:

assert isinstance( verseDataEntry, InternalBaxEntry )

marker, cleanText, extras = verseDataEntry.getMarker(), verseDataEntry.getCleanText(), verseDataEntry.getExtras()

adjustedText, originalText = verseDataEntry.getAdjustedText(), verseDataEntry.getOriginalText()

print( "marker={} cleanText={!r}{}".format( marker, cleanText, " extras={}".format( extras ) if extras else '' ) )

print( "marker={} cleanText={!r} extras={}".format( marker, cleanText, extras ) )

if adjustedText and adjustedText!=cleanText:

print( ' '\*(len(marker)+4), "adjustedText={!r}".format( adjustedText ) )

if originalText and originalText!=cleanText:

print( ' '\*(len(marker)+4), "originalText={!r}".format( originalText ) )

elif isinstance( verseDataEntry, tuple ):

marker, cleanText = verseDataEntry[0], verseDataEntry[3]

elif isinstance( verseDataEntry, str ):

if verseDataEntry=='': continue

verseDataEntry += '\n'

if verseDataEntry[0]=='\\':

marker = ''

for char in verseDataEntry[1:]:

if char!='¬' and not char.isalnum(): break

marker += char

cleanText = verseDataEntry[len(marker)+1:].lstrip()

else:

marker, cleanText = None, verseDataEntry

elif BaxOrgSysGlobals.debugFlag: halt

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " displayAppendVerse", lastParagraphMarker, haveTextFlag, marker, repr(cleanText) )

if fVM == 'Unformatted':

if marker and marker[0]=='¬': pass

elif marker in ('intro','chapters','list',): pass

else:

if isinstance( verseDataEntry, str ):

print( "marker={!r}, verseDataEntry={!r}".format( marker, verseDataEntry ) )

insertAtEnd( verseDataEntry, marker )

else:

if hadVerseText and marker in ( 's', 's1', 's2', 's3' ):

print( " Setting s mark to {}".format( nextMarkName ) )

self.textBox.mark\_set( nextMarkName, tk.INSERT )

self.textBox.mark\_gravity( nextMarkName, tk.LEFT )

print( " Inserting ({}): {!r}".format( marker, verseDataEntry ) )

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

if marker is None:

insertAtEnd( cleanText, '###' )

else: insertAtEnd( '\\{} {}'.format( marker, cleanText ), marker+'#' )

haveTextFlag = True

elif fVM == 'Formatted':

if marker.startswith( '¬' ):

if marker != '¬v': endMarkers.append( marker )

else: endMarkers = []

if marker.startswith( '¬' ):

pass

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker == 'id':

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ide','rem',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('h','toc1','toc2','toc3','cl¤',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('intro','chapters','list',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('mt1','mt2','mt3','mt4', 'imt1','imt2','imt3','imt4', 'iot','io1','io2','io3','io4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('ip','ipi','im','imi','ipq','imq','ipr', 'iq1','iq2','iq3','iq4',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('s1','s2','s3','s4', 'is1','is2','is3','is4', 'ms1','ms2','ms3','ms4', 'cl',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('d','sp',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in ('r','mr','sr',):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

insertAtEnd( cleanText, marker )

haveTextFlag = True

elif marker in BaxOrgSysGlobals.USFMParagraphMarkers:

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

lastParagraphMarker = marker

haveTextFlag = True

elif marker in ('b','ib'):

assert marker not in BaxOrgSysGlobals.USFMParagraphMarkers

assert not cleanText

if haveTextFlag: self.textBox.insert ( tk.END, '\n' )

elif marker in ('m','im'):

self.textBox.insert ( tk.END, '\n' if haveTextFlag else ' ', marker )

if cleanText:

insertAtEnd( cleanText, '\*'+marker if currentVerseFlag else marker )

lastCharWasSpace = False

haveTextFlag = True

elif marker == 'p#':

pass

elif marker == 'c':

if not firstFlag: haveC = cleanText

else: print( " Ignore C={}".format( cleanText ) )

pass

elif marker == 'c#':

if cleanText != verseKey.getBBB():

if not lastCharWasSpace: insertAtEnd( ' ', 'v-' )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

lastCharWasSpace = False

elif marker == 'v':

if cleanText != '1':

if haveTextFlag:

insertAtEnd( ' ', (lastParagraphMarker,'v-',) if lastParagraphMarker else ('v-',) )

insertAtEnd( cleanText, (lastParagraphMarker,marker,) if lastParagraphMarker else (marker,) )

insertAtEnd( '\u2009', (lastParagraphMarker,'v+',) if lastParagraphMarker else ('v+',) )

insertAtEnd( '\n', (lastParagraphMarker,'v+',) if lastParagraphMarker else ('v+',) )

haveTextFlag = True

self.lineNumber += 1

elif marker in ('v~','p~'):

needsRefreshing = appendVerseText( verseDataEntry, verseKey, currentVerseFlag, currentWordNumber=currentWordNumber, command=command )

command = None

haveTextFlag = True

else:

if BaxOrgSysGlobals.debugFlag:

logging.critical( \_("HebrewInterlinearBaxBoxAddon.displayAppendVerse (formatted): Unknown marker {!r} {!r} from {}").format( marker, cleanText, verseDataList ) )

else:

logging.critical( \_("HebrewInterlinearBaxBoxAddon.displayAppendVerse (formatted): Unknown marker {!r} {!r}").format( marker, cleanText ) )

else:

logging.critical( \_("HebrewInterlinearBaxBoxAddon.displayAppendVerse: Unknown {!r} format view mode").format( fVM ) )

if BaxOrgSysGlobals.debugFlag: halt

if lastFlag and cVM=='ByVerse' and endMarkers:

print( "endMarkers", endMarkers )

insertAtEnd( ' '+ \_("End context")+':', 'contextHeader' )

contextString, firstMarker = "", True

for someMarker in endMarkers:

print( " someMarker", someMarker )

contextString += (' ' if firstMarker else ', ') + someMarker

firstMarker = False

insertAtEnd( contextString+' ', 'context' )

if needsRefreshing: self.clearText()

else: break

def \_getBundleNumber( self, event ):

"""

触发鼠标事件

"""

xy = '@{0},{1}'.format( event.x, event.y )

print( "xy", repr(xy) )

print( "ixy", repr(self.textBox.index(xy)) )

tagNames = self.textBox.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'W' ):

bundleNumber = tagName[1:]

assert '.' in bundleNumber

print( "bundleNumber", repr(bundleNumber) )

return bundleNumber

def selectBundle( self, event ):

"""

处理鼠标左键单击动作

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.selectBundle()" )

bundleNumber = self.\_getBundleNumber( event )

if BaxOrgSysGlobals.debugFlag:

xy = '@{0},{1}'.format( event.x, event.y )

tagNames = self.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'href' ): break

tag\_range = self.tag\_prevrange( tagName, xy )

print( "tr", repr(tag\_range) )

clickedText = self.get( \*tag\_range )

print( "Clicked on {}".format( repr(clickedText) ) )

if bundleNumber:

wordNumberString, lineNumberString = bundleNumber.split( '.', 1 )

print( "select", "wn", wordNumberString, "ln", lineNumberString )

self.clearText()

self.displayAppendVerse( self.lastDAVargs[0], self.lastDAVargs[1], self.lastDAVargs[2], self.lastDAVargs[3], self.lastDAVargs[4], int(wordNumberString), None, self.lastDAVargs[7], self.lastDAVargs[8] )

def editBundle( self, event ):

"""

处理鼠标双击时间

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.editBundle()" )

bundleNumber = self.\_getBundleNumber( event )

if BaxOrgSysGlobals.debugFlag:

xy = '@{0},{1}'.format( event.x, event.y )

tagNames = self.tag\_names( xy )

print( "tn", tagNames )

for tagName in tagNames:

if tagName.startswith( 'href' ): break

tag\_range = self.tag\_prevrange( tagName, xy )

print( "tr", repr(tag\_range) )

clickedText = self.get( \*tag\_range )

print( "Clicked on {}".format( repr(clickedText) ) )

if bundleNumber:

wordNumberString, lineNumberString = bundleNumber.split( '.', 1 )

print( "edit", "wn", wordNumberString, "ln", lineNumberString )

self.clearText()

self.displayAppendVerse( self.lastDAVargs[0], self.lastDAVargs[1], self.lastDAVargs[2], self.lastDAVargs[3], self.lastDAVargs[4], int(wordNumberString), 'E', self.lastDAVargs[7], self.lastDAVargs[8] )

def doClose( self, event=None ):

"""

从GUI出调用

如果有编辑窗口，可以支持重新写入

"""

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.doClose( {} )".format( event ) )

try: self.internalBax.saveAnyChangedGlosses()

except AttributeError:

if debuggingThisModule: print( "Why is Hebrew internalBax None?" )

self.destroy()

def getBeforeAndAfterBaxData( self, newVerseKey ):

"""

返回前值，当前值；

"""

if BaxOrgSysGlobals.debugFlag:

print( "HebrewInterlinearBaxBoxAddon.getBeforeAndAfterBaxData( {} )".format( newVerseKey ) )

assert isinstance( newVerseKey, SimpleVerseKey )

BBB, C, V = newVerseKey.getBCV()

intC, intV = newVerseKey.getChapterNumberInt(), newVerseKey.getVerseNumberInt()

prevBBB, prevIntC, prevIntV = BBB, intC, intV

previousVersesData = []

for n in range( -self.parentApp.viewVersesBefore, 0 ):

failed = False

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " getBeforeAndAfterBaxData here with", n, prevIntC, prevIntV )

if prevIntV > 0: prevIntV -= 1

elif prevIntC > 0:

prevIntC -= 1

try: prevIntV = self.getNumVerses( prevBBB, prevIntC )

except KeyError:

if prevIntC != 0:

logging.error( \_("HebrewInterlinearBaxBoxAddon.getBeforeAndAfterBaxData1 failed at {} {}").format( prevBBB, prevIntC ) )

failed = True

if not failed:

if BaxOrgSysGlobals.debugFlag: print( " Went back to previous chapter", prevIntC, prevIntV, "from", BBB, C, V )

else:

try: prevBBB = self.BaxOrganisationalSystem.getPreviousBookCode( BBB )

except KeyError: prevBBB = None

if prevBBB is None: failed = True

else:

prevIntC = self.getNumChapters( prevBBB )

prevIntV = self.getNumVerses( prevBBB, prevIntC )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( " Went back to previous book", prevBBB, prevIntC, prevIntV, "from", BBB, C, V )

if prevIntC is None or prevIntV is None:

logging.error( \_("HebrewInterlinearBaxBoxAddon.getBeforeAndAfterBaxData2 failed at {} {}:{}").format( prevBBB, prevIntC, prevIntV ) )

failed = True

break

if not failed and prevIntV is not None:

print( "getBeforeAndAfterBaxData XXX", repr(prevBBB), repr(prevIntC), repr(prevIntV) )

assert prevBBB and isinstance(prevBBB, str)

previousVerseKey = SimpleVerseKey( prevBBB, prevIntC, prevIntV )

previousVerseData = self.getCachedVerseData( previousVerseKey )

if previousVerseData: previousVersesData.insert( 0, (previousVerseKey,previousVerseData,) )

nextBBB, nextIntC, nextIntV = BBB, intC, intV

nextVersesData = []

for n in range( 0, self.parentApp.viewVersesAfter ):

try: numVerses = self.getNumVerses( nextBBB, nextIntC )

except KeyError: numVerses = None

nextIntV += 1

if numVerses is None or nextIntV > numVerses:

nextIntV = 1

nextIntC += 1

nextVerseKey = SimpleVerseKey( nextBBB, nextIntC, nextIntV )

nextVerseData = self.getCachedVerseData( nextVerseKey )

if nextVerseData: nextVersesData.append( (nextVerseKey,nextVerseData,) )

verseData = self.getCachedVerseData( newVerseKey )

return verseData, previousVersesData, nextVersesData

def doBaxFind( self, event=None ):

"""

获取搜索词并进行搜索

传输搜索到的结果

注意：这个操作是在导入文件中进行操作的

所以它可以被推送到任意窗口

"""

from BaxlatorDialogs import GetBaxFindTextDialog

self.parentApp.logUsage( ProgName, debuggingThisModule, 'HebrewInterlinearBaxBoxAddon doBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.doBaxFind( {} )".format( event ) )

try: haveInternalBax = self.internalBax is not None

except AttributeError: haveInternalBax = False

if not haveInternalBax:

logging.critical( \_("No Bax to search") )

return

print( "intBib", self.internalBax )

self.BaxFindOptionsDict['currentBCV'] = self.currentVerseKey.getBCV()

gBSTD = GetBaxFindTextDialog( self, self.internalBax, self.BaxFindOptionsDict, title=\_('Find in Bax') )

if BaxOrgSysGlobals.debugFlag: print( "gBSTDResult", repr(gBSTD.result) )

if gBSTD.result:

if BaxOrgSysGlobals.debugFlag: assert isinstance( gBSTD.result, dict )

self.BaxFindOptionsDict = gBSTD.result

self.doActualBaxFind()

self.parentApp.setReadyStatus()

return tkBREAK

def doActualBaxFind( self, extendTo=None ):

"""

用于搜索函数调用

触发实际搜索

假定搜索参数已经设定

"""

from ChildWindows import FindResultWindow

self.parentApp.logUsage( ProgName, debuggingThisModule, 'HebrewInterlinearBaxBoxAddon doActualBaxFind' )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.doActualBaxFind( {} )".format( extendTo ) )

self.parentApp.setWaitStatus( \_("Searching…") )

self.textBox.update()

self.textBox.focus()

self.lastfind = key

self.parentApp.logUsage( ProgName, debuggingThisModule, ' doActualBaxFind {}'.format( self.BaxFindOptionsDict ) )

print( "bookList", repr(self.BaxFindOptionsDict['bookList']) )

bookCode = None

if isinstance( self.BaxFindOptionsDict['bookList'], str ) \

and self.BaxFindOptionsDict['bookList'] != 'ALL':

bookCode = self.BaxFindOptionsDict['bookList']

self.\_prepareInternalBax( bookCode, self.BaxFindOptionsDict['givenBax'] )

self.BaxFindOptionsDict, resultSummaryDict, findResultList = self.BaxFindOptionsDict['givenBax'].findText( self.BaxFindOptionsDict )

print( "Got findResults", findResults )

if len(findResultList) == 0:

errorBeep()

key = self.BaxFindOptionsDict['findText']

showError( self, APP\_NAME, \_("String {!r} not found").format( key if len(key)<20 else (key[:18]+'…') ) )

else:

try: replaceFunction = self.doBaxReplace

except AttributeError: replaceFunction = None

findResultWindow = FindResultWindow( self, self.BaxFindOptionsDict, resultSummaryDict, findResultList,

findFunction=self.doBaxFind, refindFunction=self.doActualBaxFind,

replaceFunction=replaceFunction, extendTo=extendTo )

self.parentApp.childWindows.append( findResultWindow )

self.parentApp.setReadyStatus()

def \_prepareInternalBax( self, bookCode=None, givenBax=None ):

"""

准备要搜索的对象或者输出检查可用对象；

注意：如果有修改，此处会被更新；

"""

logging.debug( "HebrewInterlinearBaxBoxAddon.\_prepareInternalBax()" )

if BaxOrgSysGlobals.debugFlag and debuggingThisModule:

print( "HebrewInterlinearBaxBoxAddon.\_prepareInternalBax()" )

if givenBax is None: givenBax = self.internalBax

if self.modified(): self.doSave()

if givenBax is not None:

self.parentApp.setWaitStatus( \_("Preparing internal Bax…") )

if bookCode is None:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax…") )

givenBax.load()

else:

self.parentApp.setWaitStatus( \_("Loading/Preparing internal Bax book…") )

givenBax.loadBook( bookCode )

def demo():

"""

Demo 做程序演示

"""

from tkinter import Tk

if BaxOrgSysGlobals.verbosityLevel > 0: print( ProgNameVersion )

if BaxOrgSysGlobals.verbosityLevel > 1: print( " Available CPU count =", multiprocessing.cpu\_count() )

if BaxOrgSysGlobals.debugFlag: print( "Running demo…" )

tkRootWindow = Tk()

tkRootWindow.title( ProgNameVersionDate if BaxOrgSysGlobals.debugFlag else ProgNameVersion )

HTMLTextBoxbox = HTMLTextBox( tkRootWindow )

HTMLTextBoxbox.pack()

application = Application( parent=tkRootWindow, settings=settings )

Calls to the window manager class (wm in Tk)

application.master.title( ProgNameVersion )

application.master.minsize( application.minimumXSize, application.minimumYSize )

tkRootWindow.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

from multiprocessing import freeze\_support

freeze\_support() # 支持冻结窗口多进程

parser = BaxOrgSysGlobals.setup( ProgName, ProgVersion )

BaxOrgSysGlobals.addStandardOptionsAndProcess( parser )

if 1 and BaxOrgSysGlobals.debugFlag and debuggingThisModule:

from tkinter import TclVersion, TkVersion

print( "TclVersion is", TclVersion )

print( "TkVersion is", TkVersion )

demo()

BaxOrgSysGlobals.closedown( ProgName, ProgVersion )