```tcl
#------------------------------------------------------------------------------
#  Scimitar, Source code inspector,
#     url:   http://itch.virtualave.net/scimitar/
#  Copyright (C) 2000 Agnar Renolen
#     email: agnarr@mail.virtualave.net
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
#------------------------------------------------------------------------------



namespace eval scimitar {
    global tcl_platform

    variable windowNo 0
    variable modeMap
    variable openFiles
    variable searchData
    variable lastDir [pwd]

    set searchData(last) 1.0

    variable colors
    variable fonts
    variable POPUP [expr {$tcl_platform(platform) != "macintosh"}]

    set colors(normal)   "#000000"
    set colors(reserved) "#0000ff"
    set colors(string)   "#006600"
    set colors(comment)  "#990000"

    if {$tcl_platform(platform) == "macintosh"} {
        set baseFont Monaco
    } else {
        set baseFont Courier
    }

    set fonts(normal) [font create \
            -family $baseFont \
            -size 10]
    set fonts(reserved) [font create \
            -family $baseFont \
            -size 10 \
            -weight bold]
    set fonts(comment) [font create \
            -family $baseFont \
            -size 10 \
            -slant italic]
    set fonts(string) $fonts(normal)
```

```
}

##
# NAME
#   scimitar::GetFontificationData --- retrieves information about
#   colors and fonts for standard source code components.
#
# SYNOPSIS
#   [GetFontificationData <dataVar>]
#
# DESCRIPTION
#   Sets the array named <dataVar> to contain fontification information
#   of the most common components of source code.  Each entry in the
#   array will comprise a two element list color and font ID
#   respectively.  The array will contain data for the entries:
#   \begin{description}
#    \item{[normal]} The color and font of normal text
#    \item{[reserved]} The color and font of reserved words
#    \item{[string]} The color and font of string literals
#    \item{[comment]} The color and font of comments
#   \end{description}
#   This procedure is provided to make Scimitar present source code of
#   different types of languages a uniform look and feel.
##


proc scimitar::GetFontificationData {dataVar} {
    upvar $dataVar fontData
    variable colors
    variable fonts

    foreach name [array names colors] {
        set fontData($name) [list $colors($name) $fonts($name)]
    }
}

##
# NAME
#   scimitar::DetabifyLine --- replaces tabulators with spaces
#
# SYNOPSIS
#   [DetabifyLine <string> <tabWidth>]
#
# DESCRIPTION
#   Replaces the tabulators of in <string> with spaces, adopting a
#   tabulator width of <tabWidth>. Note that <string> should only
#   contain a single line of text (ie. no new line characters)
#
#   Many source code editors inserts
#   one tabulator per indentation level, using tabulator widths of 2,
#   3 or 4 characters.  These files becomes unreadable when viewn in a
#   text editor that assumes a tabulator width of 8 characters.
#
#   Using this proc, Scimitar is enable to convert or present readably
#   source code files of any tabulator size by replacing tabulator
#   with the correct amount of spaces.
##

proc scimitar::DetabifyText { text tabSize } {
    set output ""
    set charNo 0
    foreach char [split $text {}] {
```

```
        if {$char == "\t"} {
            set numSpaces [expr $tabSize - ($charNo % $tabSize)]
            append output [string repeat " " $numSpaces]
            incr charNo $numSpaces
        } else {
            append output $char
            if {$char == "\n"} {
                set charNo 0
            } else {
                incr charNo
            }
        }
    }
    return $output
}

##
# NAME
#   scimitar::AddLineNumbers - adds line numbers to a file
#
# SYNOPSIS
#   [scimitar::AddLineNumbers <text>]
#
# DESCRIPTION
#   Prefixes each line in <text> with its line number.
##

proc scimitar::AddLineNumbers { text } {
    set lines [split $text "\n"]
    set lineCount [llength $lines]
    set fieldWidth [string length $lineCount]
    set format "%$fieldWidth"
    append format "d: %s\n"

    set output ""
    set lineNo 1
    foreach line $lines {
        append output [format $format $lineNo $line]
        incr lineNo
    }
    return $output
}



##
# NAME
#   scimitar::AssociateMode --- associates modes with a list of file
#   extensions
#
# SYNOPSIS
#   [AssociateMode <mode> <extensions>]
#
# DESCRIPTION
#   Creates an association between a set of <extensions> and a
#   <mode>.  the Modes must be implemented using a namespace named
#   <mode> in a file called [<modes>.tcl] in the [modes] directory
##

proc scimitar::AssociateMode { mode extensions } {
    variable modeMap
```

```
    foreach ext $extensions {
        set modeMap($ext) $mode
    }
}

##
# NAME
#   scimitar::NewViewer --- creates a new viewer window
#
# SYNOPSIS
#   [NewViewer <window> <windowTitle>]
#
# DESCRIPTION
#   Creates a new viewer window named <window> having the title
#   <windowTitle>.
##

proc scimitar::NewViewer {{wtitle ""}} {
    variable colors
    variable fonts
    variable openFiles
    variable windowNo

    global APP_NAME
    global RELEASE_VERSION
    global RELEASE_DATE
    global tcl_platform

    if {$tcl_platform(platform) == "macintosh"} {
        set statusFont [font create -family Geneva -size 10]
    } else {
        set statusFont [font create -family {MS Sans Serif} -size 8]
    }

    set w ".w$windowNo"
    incr windowNo

    toplevel $w \
            -menu $w.menubar

    menu $w.menubar -tearoff 0

    wm title $w $wtitle

    frame $w.edit

    text $w.edit.text \
            -width 80 \
            -height 1 \
            -xscrollcommand "$w.edit.hscroll set" \
            -yscrollcommand "$w.edit.vscroll set" \
            -wrap none \
            -font $fonts(normal) \
            -state disabled

    scrollbar $w.edit.vscroll \
            -command "$w.edit.text yview" \
            -orient vertical

    scrollbar $w.edit.hscroll \
            -command "$w.edit.text xview" \
            -orient horizontal
```

```
image create photo $w.resizeIcon -file "images/resize.gif"
label $w.edit.resize \
        -image $w.resizeIcon

label $w.edit.mode \
        -text "default mode" \
        -fg blue \
        -relief sunken \
        -bd 1

grid $w.edit.text \
        -row 0 \
        -column 0 \
        -columnspan 2 \
        -sticky news

grid $w.edit.vscroll  \
        -row 0 \
        -column 2 \
        -sticky ns

grid $w.edit.hscroll \
        -row 1 \
        -column 1 \
        -sticky we

grid $w.edit.mode \
        -row 1 \
        -column 0 \
        -ipadx 3 \
        -ipady 0

grid $w.edit.resize \
        -row 1 \
        -column 2 \
        -sticky se

# If windows, add a resize pad in lower left corner

if {$tcl_platform(platform) == "windows"} {
    bind $w.edit.resize <Button-1> \
            "scimitar::Resize $w start %X %Y"
    bind $w.edit.resize <Button1-Motion> \
            "scimitar::Resize $w move %X %Y"
    bind $w.edit.resize <ButtonRelease-1> \
            "scimitar::Resize $w end %X %Y"
}

# These two lines stalls the program on the Mac?


pack $w.edit \
        -side bottom \
        -expand true \
        -fill both

BuildMenu $w
AddWindowMenu $w

# setting up some bindings
wm protocol $w WM_DELETE_WINDOW "scimitar::CloseWindow $w"
```

```
     bind $w.edit.text <Button-1> \
            "focus $w.edit.text"



    # configuring default tags
    foreach tag {reserved comment string} {
        $w.edit.text tag configure $tag \
                -foreground $colors($tag) \
                -font $fonts($tag)
    }
    $w.edit.text tag configure search \
            -background gray

    $w.edit.text tag configure lineno \
            -background gray \
            -foreground black \
            -font $fonts(normal)

    # force tk to lay out the window so a
    # correct width can be optained
    update
    set wh [winfo vrootheight $w]
    incr wh -[winfo y $w]
    incr wh -80
    set ww [winfo reqwidth $w]

    wm geometry $w [join "$ww $wh" "x"]
    grid columnconfigure $w.edit 1 -weight 1
    grid rowconfigure $w.edit 0 -weight 1

    update

    return $w
}

##
# NAME
#   scimitar::Resize --- resizes the window
#
# SYNOPSIS
#   [Resize <window> <option> <x> <y>]
#
# DESCRIPTION
#   This proc is called when the user resizes the window by pressing
#   the resize icon on the lower right of the main window.  <option>
#   indicates differeny phases during the resize process:
#   \begin{description}
#     \item{[start]} when the user presses mouse button 1 on resize icon.
#     \item{[move]} when the user drags the resize icon
#     \item{[end]} when the user releases mouse button1 after
#     dragging.
#   \end{description}
#   <x> and <y> is the desktop coordinates of the mouse position.
##

proc scimitar::Resize {w option x y} {
    global resizeX resizeY
    global resizeW resizeH

    if {$option == "start"} {
```

```
        set resizeX $x
        set resizeY $y
        set geom [split [wm geometry $w] "=+-x"]
        set resizeW [lindex $geom 0]
        set resizeH [lindex $geom 1]
    } else {
        set width [expr $resizeW + ($x - $resizeX)]
        set height [expr $resizeH + ($y - $resizeY)]
        wm geometry $w [join [list $width $height] "x"]
    }
}

##
# NAME
#   scimitar::LoadFile --- loads a file into a window
#
# SYNOPSIS
#   [LoadFile <window> <fileName> ?<tabWidth>?]
#
# DESCRIPTION
#   Loads the contents of the file <fileName> into the window named
#   <window>.  If <tabWidth> is specified, replaces all tabulators
#   with spaces according to a specified tabulator width.
#
#   The method also checks whether there is a mode associated with the
#   file type, and installs this modes in the buffer.
##

proc scimitar::LoadFile { w fname {tabSize ""} {showLines 0}} {
    global APP_NAME
    global tcl_platform

    variable modeMap

    set in [open $fname "r"]
    set text [read $in]
    close $in

    if {$tabSize != ""} {
        set text [DetabifyText $text $tabSize]
    }

    $w.edit.text configure -state normal
    $w.edit.text delete 1.0 end
    $w.edit.text insert 1.0 $text

    set ext [file extension $fname]
    if {$tcl_platform(platform) != "unix"} {
        set ext [string tolower $ext]
    }

    if {[array names modeMap $ext] != ""} {
        set cmd "$modeMap($ext)"
        append cmd "::install $w"
        eval $cmd
        # $w.status.mode configure -text "$modeMap($ext) mode"
        $w.edit.mode configure -text "$modeMap($ext) mode"
    }
    if {$showLines} {
        set text [ShowLineNumbers $w]
    }
```

```
    $w.edit.text configure -state disabled


    # $w.status.fname configure -text "file: $fname"

    wm title $w "$APP_NAME ([file tail $fname])"
}

##
# NAME
#   scimitar::ReloadFile --- reloads a file
#
# SYNOPSIS
#   [ReloadFile <window>]
#
# DESCRIPTION
#   Reloads a file in the current buffer.  Useful if the file has
#   changed on disk.  Preserves the current tabulator width and line
#   numbering for the buffer.
##


proc scimitar::ReloadFile {w} {
    variable openFiles
    global tabSize$w
    global lineNumber$w

    set fname $openFiles($w)
    eval "LoadFile $w \"$fname\" \${tabSize$w} \${lineNumber$w}"
}




##
# NAME
#   scimitar::OpenFile --- Opens a new file
#
# SYNOPSIS
#   [OpenFile <fileName>]
#
# DESCRIPTION
#   Opens the file <fileName> in a new window.  This method first
#   calls [\see{scimitar::NewViewer}] top open a new file and then
#   [\see{scimitar::LoadFile}] to load the new file.  If the file is
#   already open in a window, then that window is brought to focus.
##

proc scimitar::OpenFile {w {fname ""}} {
    variable openFiles
    variable lastDir

    if {$fname == ""} {
        set fname [tk_getOpenFile \
                -parent $w \
                -initialdir $lastDir]
    }
    if {$fname == ""} {
        return
    }

    # check if file is already open
```

```
    foreach win [array names openFiles] {
        if {$openFiles($win) == $fname} {
            wm deiconify $win
            return
        }
    }

    set lastDir [file dirname $fname]
    if {($w == "") || ([array names openFiles $w] != "")} {
        set wname [NewViewer]
    } else {
        set wname $w
    }

    set openFiles($wname) $fname
    LoadFile $wname $fname 8


    # make the new window the active one
    wm deiconify $wname

    # the following two lines will make the text in the viewer
    # selectable.  Probably a bug in tk.
    focus $wname.edit.text
    $wname.edit.text tag add sel 1.0 1.0
    return $wname
}

##
# NAME
#   scimitar::CloseWindow
#
# SYNOPSIS
#   [CloseWindow <window>]
#
# DESCRIPTION
#   Closes the window named <window>
##

proc scimitar::CloseWindow { w } {
    variable openFiles

    # array unset openFiles $w
    if {[array names openFiles $w] != ""} {
        unset openFiles($w)
    }
    wm withdraw $w
    destroy $w

    if {[array names openFiles] == ""} {
        exit
    }
}

##
# NAME
#   scimitar::PopupWindows --- Deprecated
#
# SYNOPSIS
#   [PopupWindows <window> <x> <y>]
#
# DESCRIPTION
```

```
#   Pops up a menu listing all windows open for the current
#   application.  <window> is the active window on which the menu will
#   be displayed and <x> and <y> is the position of the mouse cursor on
#   <window>.
##

proc scimitar::PopupWindows {w x y} {
    variable openFiles

    puts Popup
    set pw $w.files
    catch {destroy $pw}
    puts "Destroy Successfull"

    menu $pw -tearoff 0

    puts "Made menu"

    foreach {window title} [array get openFiles] {
        $pw add command \
                -label $title \
                -command "wm deiconify $window"
        puts "Adding $title -> $window"
    }

    puts "Poping it up"
    tk_popup $pw $x $y
}

##
# NAME
#   scimitar::ChangeTabs --- changes the tabulator width of a text
#   buffer
#
# SYNOPSIS
#   [ChangeTabs <window> ?<tabWidth>?]
#
# DESCRIPTION
#   Reloads the file in <window> and, if <tabWidth> displayes the file
#   with a tabulator width of <tabWidth> characters, otherwisr, uses the
#   default tabulator setting of the text widget
##

proc scimitar::ChangeTabs {w {tabWidth ""}} {
    variable openFiles
    set fname $openFiles($w)
    LoadFile $w $fname $tabWidth

    global "lineNumber$w"
    eval "set show \$\{lineNumber$w\}"
    if {$show} {
        ShowLineNumbers $w
    }
}



##
# NAME
#   scimitar::ChangeLineNumbering --- hides or shows line numbers
#
# SYNOPSIS
```

```
#   [ChangeLineNumbering <window>]
#
# DESCRIPTION
#   Checks the line numbering variable associated with <window> and if
#   it is [1], calls [\see{scimitar::ShowLineNumbers}], otherwise, calls
#   [\see{scimitar::ShowLineNumbers}].  The variable determining whether
#   to show or hide line numbers is associated with the corresponding
#   menu item, and is stored in a global variable named
#   [lineNumber<window>].
##


proc scimitar::ChangeLineNumbering {w} {
    global "lineNumber$w"
    eval "set show \$\{lineNumber$w\}"
    if {$show} {
        ShowLineNumbers $w
    } else {
        HideLineNumbers $w
    }
}

##
# NAME
#   scimitar::ShowLineNumbers --- shows line numbers:
#
# SYNOPSIS
#   [ShowLineNumbers <window>]
#
# DESCRIPTION
#   Inserts a line number at the beginning of each line in the text
#   buffer of <window>.  The text inserted will have the tag [lineno]
#   associated with it.
#
# SEE ALSO
#   [\see{scimitar::HideLineNumbers}]\nl
#   [\see{scimitar::ChangeLineNumbering}]\nl
##

proc scimitar::ShowLineNumbers {w} {
    set numLines [lindex [split [$w.edit.text index end] "."] 0]

    set width [string length $numLines]
    set format " %$width"
    append format "d: "

    $w.edit.text configure -state normal
    for {set lno 1} {$lno < $numLines} {incr lno} {
        $w.edit.text insert $lno.0 [format $format $lno] lineno
    }
    $w.edit.text configure -state disabled
}

##
# NAME
#   scimitar::HideLineNumbers --- hides line numbers
#
# SYNOPSIS
#   [HideLineNumbers <window>]
#
# DESCRIPTION
#   Removes all text from the text buffer of <window> having the tag
```

```
#   [lineno] associated with it.
#
# SEE ALSO
#   [\see{scimitar::ShowLineNumbers}]\nl
#   [\see{scimitar::ChangeLineNumbering}]\nl
##

proc scimitar::HideLineNumbers {w} {
    $w.edit.text configure -state normal
    set tranges [$w.edit.text tag ranges lineno]
    foreach {idx1 idx2} $tranges {
        $w.edit.text delete $idx1 $idx2
    }
    $w.edit.text configure -state disabled
}

##
# NAME
#   scimitar::Searh --- search for text in the viewer
#
# SYNOPSIS
#   [Search <window>]
#
# DESCRIPTION
#   Brings up the search dialog and calls [\see{scimitar::DoSearch}] if
#   the user pressed the "Find" button.  Also stores search data in the
#   [searchData] cariable.
##

proc scimitar::Search {w} {
    variable searchData
    set result [searchdlg::Show $w searchData]
    if {$result == "find"} {
        DoSearch $w
    }

}

##
# NAME
#   scimitar::DoSearch --- perfoming the search
#
# SYNOPSIS
#   [DoSearch <window>]
#
# DESCRIPTION
#   Performs a search based om the information the [searchData]
#   variable, and selects the region of match if the search was
#   successul.
##

proc scimitar::DoSearch {w} {
    variable searchData

    if {$searchData(pattern) == ""} {
        return
    }

    set pattern $searchData(pattern)
    set type "-nocase"
    set end ""
```

```
    switch -exact -- $searchData(type) {
        "-word" { set pattern "\\m$pattern\\M"; set type "-regexp"  }
        "-regexp" { set type "-regexp" }
        "-exact" { set type "-exact" }
    }
    if {$searchData(wrap) == "-nowrap"} {
        set end "end"
    }

    eval "set index \[$w.edit.text search $type $searchData(direction) \
            -count count -- $pattern \"$searchData(last)\" $end\]"

    if {$index != ""} {
        $w.edit.text see $index
        text::Select $w.edit.text $index "$index + $count chars"
        set searchData(last) "$index + $count chars"
    } else {
        bell
    }
}

##
# NAME
#   scimitar::Export --- export the file into a different format
#
# SYNOPSIS
#   [Export <window> <format>]
#
# DESCRIPTION
#   Exports the current fil in one of several formats.  The following
#   formats are planned to be supported
#   \begin{itemize}
#    \item [ps] postscript
#    \item [xml] not fully supported
#    \item [html] hyper text markup language
#    \item [latex] not supported yet.
#   \end{itemize}
##

proc scimitar::Export {w format} {
    global APP_NAME
    variable openFiles
    set fname $openFiles($w)

    set fname "[lindex [split $fname .] 0].$format"

    set fname [tk_getSaveFile \
            -defaultextension ".$format" \
            -initialfile "$fname" \
            -parent $w \
            -title "Export to $format"]

    if {$fname == ""} {
        return
    }

    set output [open $fname w]
    set title [file tail $openFiles($w)]
    switch $format {
        txt  { puts $output [$w.edit.text get 1.0 end] }
        ps   { puts $output [ToPostScript $w.edit.text $title] }
        xml  { puts $output [::text::ToXML $w.edit.text] }
```

```
        html { puts $output [ToHTML $w.edit.text $title] }
        default {
            error "$APP_NAME does not export to this format ($format)"
        }
    }
    close $output
}

##
# NAME
#   scimitar::PrintFile --- Print the file
#
# SYNOPSIS
#   [PrintFile <window>]
#
# DESCRIPTION
#   Prints the file.  This procedure is not supported on all platforms:
#   \begin{description}
#    \item{windows} prints using tkprint.  There are some problems with
#    tkprint, which left me either of the following options until the
#    bug is fixed.  The current version prints the file with no syntax
#    hilighting, because tkprint have some problems when a line contains
#    tags that only span parts of the line.  Another version of tkprint
#    prints the tags correctly, but in slightly half of the correct
#    size.
#    \item{unix} Not implemented, but plan to provide printing using
#    PostScript and the lpr command.  After all, the PostScript output
#    worls pritty nice.
#    \item{macintosh} Would like to implement it in the futute.  I do
#    have a Mac at home, but I need to look into how to do it.
#   \end7description}
##


proc scimitar::PrintFile {w} {
    global tcl_platform

    switch $tcl_platform(platform) {
        windows {
            ide_print_text $w.edit.text
        }
        default {
            error "Printing on this platform is not supported"
        }
    }
}

##
# NAME
#   scimitar::GetFileName --- returns the file name of a window
#
# SYNOPSIS
#  [GetFileName <window>]
#
# DESCRIPTION
#  Returns the name of the file which is opened in <window>
##


proc scimitar::GetFileName {w} {
    variable openFiles
```

```
    return $openFiles($w)
}

##
# NAME
#   LoadModes --- Loads all modes
#
# SYNOPSIS
#   [LoadModes]
#
# DESCRIPTION
#   Loads all modes in the [modes] directory.
##

proc LoadModes {} {
    foreach fname [glob modes/*.tcl] {
        set mode [lindex [split [string tolower [file tail $fname]] "."] 0]
        append mode "::load"
        source $fname
        eval $mode
    }
}
```