```
        $env option set coupling tight
        cw_UncoupleEnvironment $env
    }

    set groupIcon(environment) $env
    $env bind changeEnvInfo                                                 \
        [list $groupIcon(class)_EnvironmentChanged $w %K]
    $env bind addEnvInfo                                                    \
        [list $groupIcon(class)_EnvironmentChanged $w %K]
    $env bind envReceived                                                   \
        [list $groupIcon(class)_EnvironmentReceived $w]
}

# ----------------------------------------------------------------------
# Because our environment is bound, we receive information about the
# widgets. Also redraw the links at this point
# ----------------------------------------------------------------------

proc groupIcon_EnvironmentReceived { w } {
    upvar #0 $w groupIcon
    $groupIcon(environment) set coords                                      \
        [$groupIcon(environment) get coords]
}

# ----------------------------------------------------------------------
# Widget update for tight coupling
# ----------------------------------------------------------------------

proc groupIcon_EnvironmentChanged { w key } {
    upvar #0 $w groupIcon

    set parts [split $key .]
    set keyItem [lindex $parts 0]

    if { $keyItem == "coords" } {
        set item [$groupIcon(environment) get $key]
        $groupIcon(-canvas) coords                                          \
            $groupIcon(canvasID) [lindex $item 0] [lindex $item 1]
    }
}

# ----------------------------------------------------------------------
# Methods for the widget
# ----------------------------------------------------------------------

proc groupIcon_Methods { w command args } {
    upvar #0 $w groupIcon
    set args [lindex $args 0]

    switch -exact $command {

        getenvironment { return $groupIcon(environment) }

        setcanvasinfo {
            set groupIcon(canvasID) [lindex $args 0]
            $groupIcon(environment) set coords                              \
                [list [lindex $args 1] [lindex $args 2]]
        }

        bindall {
            bind $w [lindex $args 0] [lindex $args 1]
            set children [info commands $w.*]
            foreach i $children {
              if { [string first ":root" $i] == -1 } {
```

```
              bind $i [lindex $args 0] [lindex $args 1]
            }
          }
      }

      setstate {
          eval "$w configure $args"
          set children [lsort [info commands $w.*]]
          foreach i $children {
            if { [string first ":root" $i] == -1 } {
              eval "$i configure $args"
            }
          }
      }

      getstate {
          set result {}
          lappend result [list $w [$w cget [lindex $args 0]]]
          set children [lsort [info commands $w.*]]
          foreach i $children {
            if { [string first ":root" $i] == -1 } {
              lappend result [list $i [$i cget [lindex $args 0]]]
            }
          }
          return $result
      }

      default { $groupIcon(rootCmd) $command $args }
    }
}

# -----------------------------------------------------------------------
# Procedures for handling the binding mechanism
# -----------------------------------------------------------------------

proc groupIcon_DragIcon { rootCanvas w x y state } {
   upvar #0 $w groupIcon

   if { $state == "relative" } {
      set cx [expr 1000 * [lindex [$rootCanvas xview] 0]]
      set cy [expr 1000 * [lindex [$rootCanvas yview] 0]]
      set x [expr $x + [winfo x $w] + $cx]
      set y [expr $y + [winfo y $w] + $cy]
   }

   #
   # Update our environment coordinates. If the icon is coupled then
   # the change will be reflected automatically on everyone else's
   # screens

   $groupIcon(environment) set coords [list $x $y]
}
```

E.2 NodeIcon hierarchical node widget

```
# -----------------------------------------------------------------------
# cw_nodeicon.tcl
#
# Latest version: 24th Jan 1996
#

# -----------------------------------------------------------------------
# nodeIcon
# This is a groupware widget consisting of an icon with some text
# underneath it
#
# It inherits the properties of a groupIcon, with the addition that it
# can be used as the basis for a node hierarchy on a canvas
# (1 parent, multiple children)
#
# It's options are:
#   parent    - widget of the parent in the hierarchy
#   children  - list of widgets of children in the hierarchy
#   line      - arrow | none, for drawing an arrow or line between links
#   linkcolor- the colour of the link line
#
# It's methods are:
#   getroot   - return the root node in the hierarchy
#   ismember  - return 1 if <item> is in the node hierarchy (here down)
#   addchild  - add a child widget to the children list
#   breaklink- break a link between this node and it's parent
#   highlight- highlight node with colour xxx
#              (option to recurse thru children)
#   tidytree  - format the nodes from the root down (intelligently...)
# ----------------------------------------------------------------------

# ----------------------------------------------------------------------
# Initialise and construct the widget
# ----------------------------------------------------------------------

proc nodeIcon { w args } {
   eval gkInt_CreateWidget $w nodeIcon nodeIconClass $args
   return $w
}

proc nodeIcon_CreateClassRec {} {
   global nodeIcon

   set nodeIcon(inherit)  {groupIcon}
   set nodeIcon(options)  {-parent -children -line -linkcolor}
   set nodeIcon(methods)  {addchild breaklink highlight tidytree
                           getroot ismember}

   set nodeIcon(-parent)    {-parent parent Parent ""}
   set nodeIcon(-children)  {-children children Children ""}
   set nodeIcon(-line)      {-line line Line arrow}
   set nodeIcon(-linkcolor) {-linkcolor linkcolor LinkColor red}

   gk_notifier breakLink
}

proc nodeIcon_InitWidgetRec { w class className args } {
   upvar #0 $w nodeIcon
   set nodeIcon(parentLinkTag) ""
   set nodeIcon(origBgColour) ""
}

proc nodeIcon_ConstructWidget { w } {
   groupIcon_ConstructWidget $w
}

# ----------------------------------------------------------------------
# Configure the widget
# ----------------------------------------------------------------------

proc nodeIcon_Config { w option args } {
   upvar #0 $w nodeIcon
   set args [lindex $args 0]
```

```
    switch -exact [string range $option 1 end] {
       parent {

          #
          # the parent node has changed, although the config variable
          # hasn't changed yet (so we can read the original value)

          set nodeIcon(parentLinkTag) "link[cw_GetCounter]"
          if { $args != "" } {
             $args addchild $w
             nodeIcon_DrawLink $w $args
          }
       }

       children {}

       line {}

       linkcolor {}

       canvas   {

          #
          # We need to update our links as we drag the icon around now
          # Also, a double-click will 'tidy' the tree from the root down

          $w bindall <B3-Motion>                                        \
             "groupIcon_DragIcon $args $w %x %y relative"
          $w bindall <Double-Button-3> "$w tidytree"
       }

       default   { groupIcon_Config $w $option $args }
    }
}

# -----------------------------------------------------------------------
# Methods for the widget
# -----------------------------------------------------------------------

proc nodeIcon_Methods { w command args } {
    upvar #0 $w nodeIcon

    set args [lindex $args 0]
    switch -exact $command   {

       getroot      { nodeIcon_GetRootNode $w }

       ismember      { nodeIcon_SeeIfMember $w [lindex $args 0] }

       addchild    { lappend nodeIcon(-children) $args }

       breaklink   { nodeIcon_BreakLink $w }

       highlight   {
          nodeIcon_Highlight $w [lindex $args 0] [lindex $args 1] }

       tidytree      { nodeIcon_TidyTree $w }

       default     { groupIcon_Methods $w $command $args }
    }
}

# -----------------------------------------------------------------------
# Create a link between this node and its parent
# -----------------------------------------------------------------------
```

```
proc nodeIcon_DrawLink { w parentW } {
   upvar #0 $w nodeIcon
   upvar #0 $parentW parentIcon

   set t $nodeIcon(parentLinkTag)
   set c1 [$nodeIcon(environment) get coords]
   set c2 [$parentIcon(environment) get coords]
   catch { $nodeIcon(-canvas) delete $t }
   if { $nodeIcon(-line) == "arrow" } {
      set mid [list [expr ([lindex $c2 0] + [lindex $c1 0]) /2]          \
         [expr ([lindex $c2 1] + [lindex $c1 1]) /2]]
      eval "$nodeIcon(-canvas) create line $c1 $mid -tag $t"
      eval "$nodeIcon(-canvas) create line $mid $c2 -arrow first         \
         -arrowshape {10 10 5} -tag $t"
   } else {
      eval "$nodeIcon(-canvas) create line $c1 $c2 -tag $t"
   }

   $nodeIcon(-canvas) itemconfigure $t                                   \
                  -fill $nodeIcon(-linkcolor) -width 2
   $nodeIcon(-canvas) bind $t <Double-Button-2>                          \
                  "nodeIcon_RequestBreakLink $w $parentW"
}

# -----------------------------------------------------------------------
# Instead of just breaking the link, fire off an event instead (which
# can be picked up and dealt with accordingly)
# -----------------------------------------------------------------------

proc nodeIcon_RequestBreakLink { nodeW parentW } {
   breakLink notify nodeIconBreakLink                                    \
      [list [list N $nodeW] [list P $parentW]]
}

# -----------------------------------------------------------------------
# Remove a link between a widget and its parent. This routine does NO
# validation checks!!
# -----------------------------------------------------------------------

proc nodeIcon_BreakLink { nodeW } {
   upvar #0 $nodeW nodeIcon
   upvar #0 $nodeIcon(-parent) parentIcon

   set children $parentIcon(-children)
   set item [lsearch $children $nodeW]
   set parentIcon(-children) [lreplace $children $item $item]
   set nodeIcon(-parent) ""
   $nodeIcon(-canvas) delete $nodeIcon(parentLinkTag)
   set nodeIcon(parentLinkTag) ""
}

# -----------------------------------------------------------------------
# Highlight a node with colour "colour" with the option of highlighting
# all its child nodes
# -----------------------------------------------------------------------

proc nodeIcon_Highlight { node colour recurse } {
   upvar #0 $node nodeIcon

   if { ($nodeIcon(origBgColour) == "") && ($colour != "reset") } {
      set nodeIcon(origBgColour) [$node getstate -background]
   }

   if { $colour == "reset" } {
      foreach i $nodeIcon(origBgColour) {
         [lindex $i 0] configure -background [lindex $i 1]
      }
   } else {
      $node setstate -background $colour
   }

   if { $recurse == "all" } {
      foreach i $nodeIcon(-children) {
         nodeIcon_Highlight $i $colour $recurse
      }
   }
}

# ---------------------------------------------------------------------
# Tidy up the tree node hierarchy
# Starting with this node, make all the nodes underneath look like a
# proper tree
# ---------------------------------------------------------------------

proc nodeIcon_TidyTree { node } {
   upvar #0 $node nodeIcon

   set widgetWidth 0
   set numNodes [nodeIcon_HowManyNodesUnderneath $node widgetWidth 1]
   set widgetWidth [cw_Min $widgetWidth 70]
   set widgetHeight [expr [winfo height $node] + 30]
   set widthAvailable [expr ($numNodes * $widgetWidth)]

   set coords [$nodeIcon(environment) get coords]
   nodeIcon_DrawTidyTree $node 0 $widthAvailable 2 $widgetWidth         \
      $widgetHeight [expr                                               \
      [lindex $coords 0] - ($widthAvailable /2) - ($widgetWidth /2)]    \
      [expr [lindex $coords 1] + $widgetHeight]
}

# ---------------------------------------------------------------------
# Intelligent counting routine for nodes underneath the current one
# e.g.  o =1,  o-o =1,  o- o o =3,  o- o o o =3,  o- o-o o o =3,
#  o- o-oo o o =5
# It will also find the widest widget of all the children
# (select level=-1 to avoid this)
# ---------------------------------------------------------------------

proc nodeIcon_HowManyNodesUnderneath { node biggestWidth level } {
   upvar #0 $node nodeIcon
   set children $nodeIcon(-children)
   set numNodes 1

   if { $level != -1 } {
      upvar $level $biggestWidth bigWidth
      set widgWidth [winfo width $node]
      if { $widgWidth > $bigWidth } { set bigWidth $widgWidth }
      incr level
   }

   set numChildren [llength $children]
   if { $numChildren > 0 } { if { [expr $numChildren % 2] == 1 } {
      incr numNodes -1 }
   }

   foreach child $children {
      incr numNodes                                                     \
         [nodeIcon_HowManyNodesUnderneath $child $biggestWidth $level]
   }
   return $numNodes
}

# ----------------------------------------------------------------------
# Recursive routine for drawing the node tree
# ----------------------------------------------------------------------

proc nodeIcon_DrawTidyTree {
                        node level widthAvailable indent wX wY rX rY } {
   upvar #0 $node nodeIcon

   set children $nodeIcon(-children)
   if { $children == "" } { return }

   set canvas $nodeIcon(-canvas)
   set totalNodes 0
   foreach child $children {
      set numNodes($child)                                             \
         [nodeIcon_HowManyNodesUnderneath $child ignore -1]
      incr totalNodes $numNodes($child)
   }

   foreach child $children {
      set spaceAvail                                                   \
         [expr ($numNodes($child) * $widthAvailable) / $totalNodes]
      set newX [expr ($spaceAvail / 2) + $indent + $rX]
      set newY [expr ($level * $wY) + $rY]

      nodeIcon_DrawTidyTree                                            \
         $child [expr $level+1] $spaceAvail $indent $wX $wY $rX $rY
      incr indent $spaceAvail

      groupIcon_DragIcon $canvas $child $newX $newY absolute
   }
}

# ----------------------------------------------------------------------
# Return the root node in the hierarchy
# ----------------------------------------------------------------------

proc nodeIcon_GetRootNode { w } {
   upvar #0 $w nodeIcon
   if { $nodeIcon(-parent) != "" } {
      return [$nodeIcon(-parent) getroot]
   }
   return $w
}

# ----------------------------------------------------------------------
# Search through the node hierarchy to see if an item is inside it
# ----------------------------------------------------------------------

proc nodeIcon_SeeIfMember { w item } {
   upvar #0 $w nodeIcon

   if { $item == $w } { return 1 }

   set children $nodeIcon(-children)

   foreach child $children {
      if { [nodeIcon_SeeIfMember $child $item] == 1 } { return 1 }
   }
   return 0
}

# ----------------------------------------------------------------------
# If the environment has changed get the node redrawn and the links
# updated
# ---------------------------------------------------------------------------

proc nodeIcon_EnvironmentChanged { w key } {
   upvar #0 $w nodeIcon

   set parts [split $key .]
   set keyItem [lindex $parts 0]
   set lastPart [lindex $parts end]

   if { $keyItem == "coords" } {
      set item [$nodeIcon(environment) get $key]
      set x [lindex $item 0]
      set y [lindex $item 1]

      $nodeIcon(-canvas) coords $nodeIcon(canvasID) $x $y
      set parent "$nodeIcon(-parent)"
      set children "$nodeIcon(-children)"

      if { $parent != "" } { nodeIcon_DrawLink $w $parent }
      foreach child $children { nodeIcon_DrawLink $child $w }

      # breakLink notify nodeIconCoordsChanged
      #    [list [list N $w] [list X $x] [list Y $y]]

   }
}

# ---------------------------------------------------------------------------
# Called whenever we first received the environment details
# ---------------------------------------------------------------------------

proc nodeIcon_EnvironmentReceived { w } {
   groupIcon_EnvironmentReceived $w
}
```

E.3 GroupTable groupware awareness widget

```
# ---------------------------------------------------------------------------
# grouptable.tcl
# Collaborwriter specialised widgets (grouptable)
#
# developed by Keith McAlpine
# started 19th November 1994
# rewritten to use gk_views constructs
#
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
# Basic design philosophy:
#
# grouptable widget
# This widget consists of a canvas item consisting of a 'table' around
# which participants can  be placed.  This can be used for showing who
# is currently participating in a conference (using the user colours
# for each person as a representation).
#
# <widget> adduser                - adds a participant to the table
# <widget> deleteuser             - removes a participant from the table
# <widget> attributechanged       - changes colour of participant <uniq id>
#
# Configuration options:
# -tablecolour <newcolour>        - changes the colour of the table
#                                  (default: brown)
# <-personsize <value | auto> - set up size of the people in the widget
#                                  (default: auto)
#
# You can also use the standard canvas configure commands eg. config
# bg to change the background (default: DarkSalmon).
# The width/height values can also be changed so that the canvas items
# get resized automatically (or not, as set)
# -----------------------------------------------------------------------

# miscellaneous utility routines

proc min { a b } { return [expr ($a < $b) ? $a : $b] }
proc max { a b } { return [expr ($a < $b) ? $b : $a] }

# -----------------------------------------------------------------------
# Routines used by the ClassBuilder routines in Groupkit for the easy
# creation of widgets
# -----------------------------------------------------------------------

proc grouptable {w args} {

   #
   # Create a grouptable

   eval gkInt_CreateWidget $w grouptable grouptableClass $args
   return $w
}

proc grouptable_CreateClassRec {} {
   global grouptable

   #
   # Setup the class record for the grouptable.
   # It inherits all the commands/options from the canvas

   set grouptable(inherit) {canvas}
   set grouptable(options) {-tablecolour -personsize -scrollid}
   set grouptable(methods) {adduser attributechanged
                             deleteuser moveuser }

   set grouptable(-tablecolour) {
                 -tablecolour tableColour TableColour brown}
   set grouptable(-personsize) {-personsize personSize PersonSize auto}
   set grouptable(-scrollid)   {-scrollid scrollId ScrollId none}
}

proc grouptable_ConstructWidget { w } {
   upvar #0 $w data

   canvas $data(widget).c -bg DarkSalmon
   pack $data(widget).c
}

# -----------------------------------------------------------------------
# Routines used internally by the grouptable widget
# -----------------------------------------------------------------------

proc grouptable_InitWidgetRec {w class className args} {
   upvar #0 $w data

   #
   # Initialize all the variables needed for the grouptable

   set data(local) [users local.usernum]
   set data(table) ""
```

```
    set data(widget) $w
    set data(canvasItems) ""
    set data(userlist) {}
    set data(colourlist) {}
}

# ----------------------------------------------------------------------------
# Routine called on the result of a <widget> configure xxx command
# ----------------------------------------------------------------------------

proc grouptable_Config {w option args} {
   upvar #0 $w data
   set args [lindex $args 0]

   switch -exact [string range $option 1 end] {
      tablecolour {
         set data(-tablecolour) $args
         $data(widget).c itemconfigure $data(table)                      \
            -fill $data(-tablecolour)
      }

      personsize {
         set data(-personsize) $args
         grouptable_DrawPeople $w
      }

      width -
      height {
         $data(widget).c config $option $args
         grouptable_DrawPeople $w
      }

      scrollid {
         set data(-scrollid) $args
         set data(userlist) {}
         set data(colourlist) {}
         if { $args != "none" } {
            gk_views $w $data(-scrollid)
            gk_viewsSetCoords                                            \
               $data(-scrollid) $data(local) [list 0 0 1 1]
            gk_viewsSetAttribute $data(-scrollid)                        \
               $data(local) color [userprefs color]
         }
      }

      default {
         $data(widget).c config $option $args
      }
   }
}

# ----------------------------------------------------------------------------
# Routine called on result of a <widget> <command> xxx command
# ----------------------------------------------------------------------------

proc grouptable_Methods {w command args} {
   upvar #0 $w data
   set args [lindex $args 0]

   cw_Debug "grouptable method: $command - $args - $w" white

   switch -exact $command {

      adduser {
         if { [llength $args] == 2 } {
            set idx [lsearch $data(userlist) [lindex $args 0]]
            if { $idx == -1 } {
               lappend data(userlist) [lindex $args 0]
               lappend data(colourlist) [lindex $args 1]
            }
         }
         grouptable_DrawPeople $w
      }

      deleteuser {
         if { [llength $args] == 2 } {
            set idx [lsearch $data(userlist) [lindex $args 1]]
            if { $idx != -1 } {
               set data(userlist) [lreplace $data(userlist) $idx $idx]
               set data(colourlist)                                      \
                  [lreplace $data(colourlist) $idx $idx]
            }
         }
         grouptable_DrawPeople $w
      }

      attributechanged {
         if { [lindex $args 2] == "color" } {
            set idx                                                      \
               [cw_ListSearch $data(canvasItems) 0 [lindex $args 1]]
            if { $idx != -1 } {
               $data(widget).c itemconfigure                             \
                  [lindex [lindex $data(canvasItems) $idx] 1]            \
                  -fill [lindex $args 3]
            }
         }
      }

      moveuser {}

      default   {
         $data(widget).c $command $args
         if [catch "$data(widget).v $command $args"] {}
      }
   }
}

# ------------------------------------------------------------------------
# Routine to actually draw the items in the grouptable widget
# ------------------------------------------------------------------------

proc grouptable_DrawPeople { w } {
   upvar #0 $w data

   set width [lindex [$data(widget).c configure -width] 4]
   set height [lindex [$data(widget).c configure -height] 4]
   set personsize [lindex [$data(widget) configure -personsize] 4]

   set minsize [min $width $height]
   set middlex [expr $width / 2]
   set middley [expr $height / 2]

   #
   # now work out the dimensions of the table, people, distance from
   # table (t=1.5p, d=2p, p=auto | value)

   if { $personsize == "auto" } {
      set personsize [max [expr $minsize / 6] 1]
   }
   set tablesize [max [expr $personsize * 1.5] 2]
   set distance [max [expr $personsize * 2] 1]
```

```
    #
    # first get the central table drawn in the middle of the cvas widget

    if { $data(table) != "" } { $data(widget).c delete $data(table) }
    set data(table) [$data(widget).c create oval                              \
       [expr $middlex -($tablesize/2)] [expr $middley - ($tablesize/2)] \
       [expr $middlex +($tablesize/2)] [expr $middley + ($tablesize/2)] \
       -fill [lindex [$w configure -tablecolour] 4]]

    $data(widget).c bind $data(table) <1>                                     \
       "_groupTable_PopupAllMenu $data(widget).c $w %X %Y"
    $data(widget).c bind $data(table) <B1-ButtonRelease>                      \
       "destroy $data(widget).c.popup"

    #
    # now get rid of any people currently displayed on the conf table

    foreach id $data(canvasItems) {
       $data(widget).c delete [lindex $id 1]
    }
    set data(canvasItems) ""

    #
    # and get them redrawn here

    if { $data(-scrollid) == "none" } {
       set users $data(userlist)
    } else {
       set users [gk_viewsUsers $data(-scrollid)]
    }

    cw_Debug "users for $w ( $data(-scrollid) ) are: $users" purple

    set numpeople [llength $users]
    if { $numpeople > 0 } { set step [expr 360 / $numpeople] }
    set place 0
    set listpos 0

    set repack {}
    foreach person $users {

       if { $data(-scrollid) == "none" } {
          set colour [lindex $data(colourlist) $listpos]
          incr listpos
       } else {
          set colour                                                          \
             [gk_viewsGetAttribute $data(-scrollid) $person color]
       }
       if { $colour == "" } { set colour black; lappend repack $person }

       set angle [expr ($place * 3.1415927) / 180]
       set x [expr $middlex -($distance *sin($angle)) - ($personsize /2)]
       set y [expr $middley -($distance *cos($angle)) - ($personsize /2)]

       set id [$data(widget).c create oval $x $y [expr $x +$personsize] \
          [expr $y + $personsize] -fill $colour]
       _groupTable_BindPopup $data(widget).c $id $person $data(-scrollid)
       lappend data(canvasItems) [list $person $id]
       incr place $step
    }
}

# -----------------------------------------------------------------
# Set the bindings for the popup menu
# -----------------------------------------------------------------
```

```
proc _groupTable_BindPopup { w id user viewID } {
   $w bind $id <1> "_groupTable_PopupMenu $w $user 0 %X %Y $viewID"
   $w bind $id <B1-ButtonRelease> "destroy $w.popup"
}

# ----------------------------------------------------------------------
# Have little popup menus showing the user name underneath a grouptable
# colour circle
# ----------------------------------------------------------------------

proc _groupTable_PopupMenu {w user showcolour x y scrollID} {
   catch { destroy $w.popup }
   menu $w.popup -tearoff 0

   if { $user == [users local.usernum] } {
      $w.popup add command -label "This is you"
   } else {
      $w.popup add command -label [users remote.$user.username]
   }

   if { $showcolour } {
      set colour [gk_viewsGetAttribute $scrollID $user color]
      $w.popup add command -background $colour
   }

   $w.popup post $x $y
   grab $w.popup
}

# ----------------------------------------------------------------------
# If you click on the table, a popup shows all the participants in the
# node
# ----------------------------------------------------------------------

proc _groupTable_PopupAllMenu { w vars x y } {
   upvar #0 $vars data

   catch { destroy $w.popup }
   menu $w.popup -tearoff 0

   set users $data(canvasItems)
   if { [llength $users] == 0 } {
      $w.popup add command -label "Nobody"
   } else {
      foreach i $users {
         set id [lindex $i 0]
         if { $id == [users local.usernum] } {
            $w.popup add command -label "Yourself"
         } else {
            $w.popup add command -label [users remote.$id.username]
         }
      }
   }
   $w.popup post $x $y
   grab $w.popup
}
```

E.4 PictureBar groupware awareness widget

```
# ----------------------------------------------------------------------
# picturebar.tcl
# Collaborwriter specialised widgets (picturebar)
```

```
#
# developed by Keith McAlpine
#
# ----------------------------------------------------------------------

# ----------------------------------------------------------------------
# Basic design philosophy:
#
# picturebar widget
# This widget consists of a row of users who are inside a particular
# widget
#
# <widget> configure -command   - command to execute on Double-1 click
#
# <widget> adduser              - adds a participant to the row
# <widget> deleteuser           - removes a participant from the row
# <widget> attributechanged     - changes icon of participant <unique id>
# <widget> getselection           - return list of selected <unique id>s
#
# ----------------------------------------------------------------------

# miscellaneous utility routines

proc min { a b } { return [expr ($a < $b) ? $a : $b] }
proc max { a b } { return [expr ($a < $b) ? $b : $a] }

# ----------------------------------------------------------------------
# Routines used by the ClassBuilder routines in Groupkit for the easy
# creation of widgets
# ----------------------------------------------------------------------

proc picturebar {w args} {

   # Create a picturebar

   eval gkInt_CreateWidget $w picturebar picturebarClass $args
   return $w
}

proc picturebar_CreateClassRec {} {
   global picturebar

   # Setup the class record for the picturebar.
   # It inherits all the commands/options from the canvas

   set picturebar(inherit) {label}
   set picturebar(options) {-scrollid -command}
   set picturebar(methods) {adduser attributechanged deleteuser
                            moveuser getselection}
   set picturebar(-scrollid) {-scrollid scrollId ScrollId picturebar}
   set picturebar(-command)  {-command command Command ""}
}

proc picturebar_ConstructWidget { w } {
   upvar #0 $w data
}

proc picturebar_DestroyWidget { w } {
   upvar #0 $w data

   if { $data(-scrollid) != "none" } {
      gk_viewsDelete $data(-scrollid) [users local.usernum]
   }
}

# ----------------------------------------------------------------------
# Routines used internally by the picturebar widget
# ----------------------------------------------------------------------
```

```
proc picturebar_InitWidgetRec {w class className args} {
   upvar #0 $w data

   #
   # Initialize all the variables needed for the picturebar

   set data(local) [users local.usernum]
   set data(widget) $w
   set data(userlist) {}
   set data(iconlist) {}
   set data(selection) {}
}

# ---------------------------------------------------------------------
# Routine called on the result of a <widget> configure xxx command
# ---------------------------------------------------------------------

proc picturebar_Config {w option args} {
   upvar #0 $w data
   set args [lindex $args 0]

   switch -exact [string range $option 1 end] {
      scrollid {
         set data(-scrollid) $args
         set data(userlist) {}
         set data(iconlist) {}
         set data(selection) {}
            if { $args != "none" } {
               gk_views $w $data(-scrollid)
               gk_viewsSetCoords $data(-scrollid) $data(local)         \
                  [list 0 0 1 1]
               gk_viewsSetAttribute $data(-scrollid)                   \
                  $data(local) color [userprefs color]
               gk_viewsSetAttribute $data(-scrollid)                   \
                  $data(local) iconname [users local.iconpict]
            }
         }

      default {}
   }
}


# ---------------------------------------------------------------------
# Routine called on result of a <widget> <command> xxx command
# ---------------------------------------------------------------------

proc picturebar_Methods {w command args} {
   upvar #0 $w data
   set args [lindex $args 0]

   cw_Debug "picturebar method: $command - $args - $w" white

   switch -exact $command {

      adduser {
         if { $data(-scrollid) != "" } {
            set userid [lindex $args 1]
            set idx [lsearch $data(userlist) $userid]
            if { ($idx == -1) && ($userid != "") } {
               lappend data(userlist) $userid
               set img [gk_viewsGetAttribute $data(-scrollid)          \
                  $userid iconname]
               lappend data(iconlist) $img
            lappend data(selection) 0
            cw_CreateImage [image types] $img
            set icn $w.$userid
            #
            # if our picture isn't displayed draw it and set up some
            # default bindings for it
            if { ![winfo exists $icn] } {
               frame $icn
               label $icn.img -image $img
               pack $icn -side left -fill y
               pack $icn.img -fill both -anchor s
               bind $icn.img <1>                                      \
                  "_picturebar_ToggleSelection $w $userid"
               bind $icn.img <Double-1>                               \
                  "_picturebar_DoCommand $w $userid"
               bind $icn.img <2> "_groupTable_PopupMenu               \
                  $icn.img $userid 1 %X %Y $data(-scrollid)"
               bind $icn.img <B2-ButtonRelease>                       \
                  "destroy $icn.img.popup"
            }
         }
      }
   }

   deleteuser {
      if { [llength $args] == 2 } {
         set idx [lsearch $data(userlist) [lindex $args 1]]
         if { $idx != -1 } {
            destroy $w.[lindex $args 1]
            cw_DeleteImage [lindex $data(iconlist) $idx]
            set data(userlist) [lreplace $data(userlist) $idx $idx]
            set data(iconlist) [lreplace $data(iconlist) $idx $idx]
            set data(selection) [lreplace $data(selection) $idx $idx]
         }
      }
   }

   attributechanged {
      set attr [lindex $args 2]
      set userid [lindex $args 1]
      if { $attr == "iconname" } {
         set img [lindex $args 3]
         set idx [lsearch $data(userlist) $userid]
         if { $idx != -1 } {
            set oldimg [lindex $data(iconlist) $idx]
            cw_CreateImage [image types] $img
            $w.$userid.img configure -image $img
            cw_DeleteImage $oldimg
            set data(iconlist)                                        \
               [lreplace $data(iconlist) $idx $idx $img]
         }
      }
   }

   moveuser {}

   default   {}
  }
}
# -----------------------------------------------------------------------
# Toggle the selection of a specific picturebar icon element
# -----------------------------------------------------------------------
```

```
proc _picturebar_ToggleSelection { w userid } {
   upvar #0 $w data

   set idx [lsearch $data(userlist) $userid]
   if { $idx != -1 } {
      set state [lindex $data(selection) $idx]
      set state [expr !$state]
      if { $state == 0 } {
         $w.$userid.img configure -background #d9d9d9
      } else {
         $w.$userid.img configure -background yellow
      }
      set data(selection) [lreplace $data(selection) $idx $idx $state]
   }
}

# ----------------------------------------------------------------------
# Execute an appropriate command when double-clicking on a picturebar
# ----------------------------------------------------------------------

proc _picturebar_DoCommand { w userid } {
   upvar #0 $w data

   if { $data(-command) != "" } {
      set selectedIDs {}
      for { set i 0 } { $i < [llength $data(selection)] } { incr i } {
         if { [lindex $data(selection) $i] == 1 } {
            lappend selectedIDs [lindex $data(userlist) $i]
         }
      }

      #
      # make sure the item being double-clicked on is also selected

      set idx [lsearch $data(userlist) $userid]
      if { [lindex $data(selection) $idx] == 0 } {
         lappend selectedIDs $userid
         _picturebar_ToggleSelection $w $userid
      }

      eval $data(-command) $w $userid $selectedIDs
   }
}
```

E.5 AttrPictureBar extension to the PictureBar widget

```
# ----------------------------------------------------------------------
# attrpicturebar.tcl
# Collaborwriter specialised widgets (attrpicturebar)
#
# developed by Keith McAlpine
#
# ----------------------------------------------------------------------

# ----------------------------------------------------------------------
# Basic design philosophy:
#
# attrpicturebar widget
# This widget is an override of the picturebar widget. It is used to
# provide access rights underneath the picture element. If clicked on,
# these changes are passed as events back to the calling application
# (although only to one calling app)
```

```
#
# ------------------------------------------------------------------------

# ------------------------------------------------------------------------
# Utility routine for initialising the image pictures - call once in
# the initialisations of the calling program
# ------------------------------------------------------------------------

proc attrpicturebar_Init {} {
   set d "/phd/gk/cw/bitmaps"

   if {![cw_CreateNamedImage [image types] $d/readicon readicon]} {
      error "No readicon icon"}
   if {![cw_CreateNamedImage [image types]                              \
      $d/commenticon commenticon]} {error "No commenticon icon"}
   if {![cw_CreateNamedImage [image types] $d/writeicon writeicon]} {
      error "No writeicon icon"}
   if {![cw_CreateNamedImage [image types] $d/noreadicon noreadicon]} {
      error "No noreadicon icon"}
   if {![cw_CreateNamedImage [image types]                              \
      $d/nocommenticon nocommenticon]} {error "No nocommenticon icon"}
   if {![cw_CreateNamedImage [image types]                              \
      $d/nowriteicon nowriteicon]} { error "No nowriteicon icon" }
}

# ------------------------------------------------------------------------
# Routines used by the ClassBuilder routines in Groupkit for the easy
# creation of widgets
# ------------------------------------------------------------------------

proc attrpicturebar {w args} {

   # Create a attrpicturebar

   eval gkInt_CreateWidget $w attrpicturebar attrpicturebarClass $args
   return $w
}

proc attrpicturebar_CreateClassRec {} {
   global attrpicturebar

   # Setup the class record for the attrpicturebar.
   # It inherits all the commands/options from the canvas

   set attrpicturebar(inherit) {picturebar}
   set attrpicturebar(options) {-scrollid -canchange}
   set attrpicturebar(methods) { adduser attributechanged deleteuser
                                 moveuser }

   set attrpicturebar(-scrollid)  {-scrollid scrollId ScrollId none}
   set attrpicturebar(-canchange) {-canchange canChange CanChange no}
}

proc attrpicturebar_ConstructWidget { w } {
}

# ------------------------------------------------------------------------
# Routines used internally by the attrpicturebar widget
# ------------------------------------------------------------------------

proc attrpicturebar_InitWidgetRec {w class className args} {
   upvar #0 $w data

   # Initialize all the variables needed for the attrpicturebar

   picturebar_InitWidgetRec $w $class $className [lindex $args 0]
}
```

```
proc attrpicturebar_DestroyWidget { w } {
   puts "destroying attrpicturebar"
   picturebar_DestroyWidget $w
}

# -----------------------------------------------------------------------
# Routine called on the result of a <widget> configure xxx command
# -----------------------------------------------------------------------

proc attrpicturebar_Config {w option args} {
   upvar #0 $w data

   set args [lindex $args 0]
   picturebar_Config $w $option $args

   switch -exact [string range $option 1 end] {

      canchange {
         if { [lindex $args 0] == 0 } {
            set state disabled } else { set state active }

         set widgets [winfo children $w]
         foreach i $widgets {
            $i.access.r configure -state $state
            $i.access.c configure -state $state
            $i.access.w configure -state $state
         }
      }

      default {}
   }
}


# -----------------------------------------------------------------------
# Routine called on result of a <widget> <command> xxx command
# -----------------------------------------------------------------------

proc attrpicturebar_Methods {w command args} {
   upvar #0 $w data

   set args [lindex $args 0]
   picturebar_Methods $w $command $args

   cw_Debug "attrpicturebar method: $command - $args - $w" white

   switch -exact $command {
      adduser {
         set userid [lindex $args 1]
         set idx [lsearch $data(userlist) $userid]
         if { $idx != -1 } {
            set icn $w.$userid
            if { [winfo exists $icn.access] == 0 } {
            if { $data(-canchange) == 0 } {
               set state disabled } else { set state active }

            frame $icn.access -relief groove -bd 2
            radiobutton $icn.access.r -selectimage readicon -bd 1        \
               -image noreadicon -padx 0 -pady 0 -indicatoron 0          \
               -variable access-$userid -value r -state $state
            radiobutton $icn.access.c -selectimage commenticon -bd 1     \
               -image nocommenticon -padx 0 -pady 0 -indicatoron 0       \
               -variable access-$userid -value c -state $state
            radiobutton $icn.access.w -selectimage writeicon -bd 1       \
               -image nowriteicon -padx 0 -pady 0 -indicatoron 0         \
                -variable access-$userid -value w -state $state

                pack $icn.access -side bottom -anchor s
                pack $icn.access.r $icn.access.c $icn.access.w             \
                   -side left -anchor s
              }
            }
          }

          default   {}
      }
}
```

E.6 IBISicon groupware argumentation widget

```
# ---------------------------------------------------------------------
# cw_ibisicon.tcl
#
# Latest version: 24th Jan 1996
#


# ---------------------------------------------------------------------
# ibisIcon
# This is a groupware widget consisting of an icon with some text
# inside it, with colours underneath for identifying agreement or not.
# These colours can be changed by the user to yes | no | undecided
#
# It inherits the properties of a groupIcon, with the addition that it
# can be used as the basis for a node graph on a canvas
# (no parent, multiple attachments)
#
# Being based on the IBIS ideas, the link lines can be of different
# types depending on the relationship with the other nodes
#
# It's options are:
#   children - list of widgets of children in the hierarchy
#   line     - generalises | specialises | replaces | questions |
#              suggestedby| supports| objects| responds| custom| none
#   linkstyle- text | icon
#   entity   - issue | position | argument
#   strict   - yes | no - is the ibis structure strictly enforced ?
#   showvotes- yes | no - whether to display the voting widget part
#   menucmd  - command to call on a popup menu selection
#
# It's methods are:
#   breaklink- break a link between this node and it's parent
#   highlight- highlight node with colour xxx
#              (option to recurse thru children)
#
#   changeattachments   - change what nodes are attached to this entity
# ---------------------------------------------------------------------


# ---------------------------------------------------------------------
# Initialise and construct the widget
# ---------------------------------------------------------------------

proc ibisIcon { w args } {
   eval gkInt_CreateWidget $w ibisIcon ibisIconClass $args
   return $w
}
```

```
proc ibisIcon_CreateClassRec {} {
   global ibisIcon

   set ibisIcon(inherit)   {groupIcon}
   set ibisIcon(options)   {-entity -text -strict -showvotes
                            -menucmd -linkstyle -scrollid}
   set ibisIcon(methods)   {breaklink highlight getattachments
                            changeattachments adduser canattach
                            attributechanged deleteuser moveuser
                            addcustom copycustom}

   set ibisIcon(-entity)      {-entity entity Entity issue}
   set ibisicon(-text)        {-text text Text ""}
   set ibisIcon(-menucmd)     {-menucmd menuCmd MenuCmd ibisIcon_Menu}
   set ibisIcon(-linkstyle)   {-linkstyle linkStyle LinkStyle text}
   set ibisIcon(-scrollid)    {-scrollid scrollId ScrollId ibisIcon}
   set ibisIcon(-strict)      {-strict strict Strict yes}
   set ibisIcon(-showvotes)   {-showvotes showVotes ShowVotes yes}

   gk_notifier breakLink
}

proc ibisIcon_InitWidgetRec { w class className args } {
   upvar #0 $w ibisIcon

   set ibisIcon(origBgColour) ""
   set ibisIcon(attachments) {}
   set ibisIcon(local) [users local.usernum]
   set ibisIcon(userlist) {}
   set ibisIcon(colourlist) {}
   set ibisIcon(choicelist) {}
}

proc ibisIcon_ConstructWidget { w } {
   groupIcon_ConstructWidget $w

   after 1 "pack forget $w.icontext"
   frame $w.choices -background #d9d9d9
   pack $w.choices -fill x -expand yes
}

# -----------------------------------------------------------------------
# Configure the widget
# -----------------------------------------------------------------------

proc ibisIcon_Config { w option args } {
   upvar #0 $w data
   set args [lindex $args 0]

   switch -exact [string range $option 1 end] {
      scrollid {
         set data(-scrollid) $args
         set data(userlist) {}
         set data(colourlist) {}
         set data(choicelist) {}
         if { $args != "none" } {
            gk_views $w $data(-scrollid)
            gk_viewsSetCoords $data(-scrollid) $data(local) {0 0 1 1}
            gk_viewsSetAttribute $data(-scrollid) $data(local)          \
               color [userprefs color]
            gk_viewsSetAttribute $data(-scrollid) $data(local)          \
               choice undecided

         }
      }
      linkstyle {
         set $data(-linkstyle) $args

         #
         # kick the node into redrawing all its links

         set children "$data(attachments)"
         foreach child $children {
            if { [lindex $child 3] == "end" } {
               ibisIcon_DrawLink $args $w                                    \
                  [lindex $child 0] [lindex $child 1] [lindex $child 2]
            } else {
               ibisIcon_DrawLink $args [lindex $child 0]                     \
                  $w [lindex $child 1] [lindex $child 2]
            }
         }

         #$data(environment) set coords [$data(environment) get coords]
      }

      menucmd {
         set $data(-menucmd) $args
         ibisIcon_BuildPopup $w $args }

      entity {
         set $data(-entity) $args
         ibisIcon_BuildPopup $w $data(-menucmd) }

      text {
         if { $data(environment) != "" } {
            $data(environment) set ibistext $args
         }
      }

      canvas   {

         #
         # We need to update our links as we drag the icon around now
         # Also, a double-click will 'tidy' the tree from the root down

         $w bindall <B3-Motion>                                              \
            "groupIcon_DragIcon $args $w %x %y relative"
         $w bindall <2> "ibisIcon_PopupMenu $w %X %Y"
         $w bindall <Double-Button-1> "ibisIcon_NodeDialog $w"
         $w bindall <B2-ButtonRelease> "destroy $w.popup"
      }

      showvotes {
         if { $args == "yes" } { pack $w.choices
         } else { pack forget $w.choices }
      }

      strict { ibisIcon_BuildPopup $w $data(-menucmd) }

      default  { groupIcon_Config $w $option $args }
   }
}

# ---------------------------------------------------------------------------
# Methods for the widget
# ---------------------------------------------------------------------------

proc ibisIcon_Methods { w command args } {
   upvar #0 $w data
```

```
set args [lindex $args 0]

switch -exact $command    {
   adduser     {
      if { [llength $args] == 2 } {
         set idx [lsearch $data(userlist) [lindex $args 0]]
         if { $idx == -1 } {
            lappend data(userlist) [lindex $args 1]
            lappend data(colourlist) black
            lappend data(choicelist) undecided
         }
      }
      if { [winfo exists $w.choices.[lindex $args 1]] == 0 } {
         choicebutton $w.choices.[lindex $args 1]                        \
            -currentvalue undecided -callcmd ibisIcon_ChoiceClick
         pack $w.choices.[lindex $args 1] -side left
         if { [lindex $args 1] != $data(local) } {
            $w.choices.[lindex $args 1] configure                        \
               -disabledforeground black -state disabled
         }
      }
   }

   deleteuser   {
      if { [llength $args] == 1 } {
         set i [lsearch $data(userlist) [lindex $args 0]]
         if { $i != -1 } {
            set data(userlist) [lreplace $data(userlist) $i $i]
            set data(colourlist) [lreplace $data(colourlist) $i $i]
            set data(choicelist) [lreplace $data(choicelist) $i $i]
         }
      }
      if { [winfo exists $w.choices.[lindex $args 0]] } {
         destroy $w.choices.[lindex $args 0]
      }
   }

   attributechanged {
      set state [$w.choices.[lindex $args 1] cget -state]
      puts "attr changed: [users local.usernum] - $args $state"
      if { [lindex $args 2] == "color" } {
         $w.choices.[lindex $args 1]                                     \
            configure -state normal -background [lindex $args 3]         \
            -activebackground [lindex $args 3] -state $state
      } elseif { [lindex $args 2] == "choice" } {
         $w.choices.[lindex $args 1] configure -state normal
         $w.choices.[lindex $args 1] configure                           \
            -currentvalue [lindex $args 3] -state $state
      }
   }

   moveuser      {}

   getattachments { return $data(attachments) }

   canattach { ibisIcon_CanAttach $w $args }

   breaklink   { ibisIcon_BreakLink $w }

   highlight   {
      ibisIcon_Highlight $w [lindex $args 0] [lindex $args 1] }
      changeattachments { ibisIcon_ChangeAttachments $w $args }

      bindall      {
         bind $w [lindex $args 0] [lindex $args 1]
         set children [info commands $w.*]
         foreach i $children {
            if { ([string first "choices." $i] == -1) &&                    \
                                 ([string first ":root" $i] == -1) } {
               bind $i [lindex $args 0] [lindex $args 1]
            }
         }
      }

      setstate      {
         eval "$w configure $args"
         set children [lsort [info commands $w.*]]
         foreach i $children {
            if { ([string first "choices." $i] == -1) &&                    \
                                 ([string first ":root" $i] == -1) } {
               eval "$i configure $args"
            }
         }
      }

      getstate      {
         set result {}
         lappend result [list $w [$w cget [lindex $args 0]]]
         set children [lsort [info commands $w.*]]
         foreach i $children {
            if { ([string first "choices." $i] == -1) &&                    \
                                 ([string first ":root" $i] == -1) } {
               lappend result [list $i [$i cget [lindex $args 0]]]
            }
           }
           return $result
      }

      addcustom {
         set custom [$data(environment) get custom-linktypes]
         set custom [concat $custom [list $args]]
         $data(environment) set custom-linktypes $custom
      }

      copycustom {
         set otherWidget $args
         set custom [$data(environment) get custom-linktypes]
         ibisIcon_DoCopy $otherWidget [list $custom]
      }

      default     { groupIcon_Methods $w $command $args }
   }
}

# --------------------------------------------------------------------------
# Put the custom widgets from 1 node into its children
# --------------------------------------------------------------------------

proc ibisIcon_DoCopy { w info } {
   upvar #0 $w data
   $data(environment) set custom-linktypes [lindex $info 0]
   ibisIcon_BuildPopup $w $data(-menucmd)
}
```

```
# ----------------------------------------------------------------------
# Build up a context-sensitive popup menu
# ----------------------------------------------------------------------

proc ibisIcon_BuildPopup { w cmd } {
   upvar #0 $w data

   #
   # depending on the entity there's a different popup menu

   catch { destroy $w.popup }
   menu $w.popup -tearoff 0
   switch -exact $data(-entity) {
      issue {
         $w.popup add command -label "Issue generalises"                    \
            -command "eval $cmd $w issue generalises"
         $w.popup add command -label "Issue specialises"                    \
            -command "eval $cmd $w issue specialises"
         $w.popup add command -label "Issue replaces"                       \
            -command "eval $cmd $w issue replaces"
         $w.popup add command -label "Issue questions"                      \
            -command "eval $cmd $w issue questions"
         $w.popup add command -label "Issue is suggested by"                \
            -command "eval $cmd $w issue suggestedby"
         $w.popup add separator
         $w.popup add command -label "Position responds to"                 \
            -command "eval $cmd $w position responds"
         $w.icon configure -bd 4 -relief groove                             \
            -font "-*-helvetica-bold-r-*-*-10-*"
      }
      position {
         $w.popup add command -label "Issue questions"                      \
            -command "eval $cmd $w issue questions"
         $w.popup add command -label "Issue is suggested by"                \
            -command "eval $cmd $w issue suggestedby"
         $w.popup add separator
         $w.popup add command -label "Argument supports"                    \
            -command "eval $cmd $w argument supports"
         $w.popup add command -label "Argument objects to"                  \
            -command "eval $cmd $w argument objects"
         $w.icon configure -bd 1 -relief flat                               \
      }
      argument {
         $w.popup add command -label "Issue questions"                      \
            -command "eval $cmd $w issue questions"
         $w.popup add command -label "Issue is suggested by"                \
            -command "eval $cmd $w issue suggestedby"
         $w.icon configure -bd 1 -relief flat                               \
            -font "-*-helvetica-medium-o-*-*-10-*"

      }
   }

   #
   # if this ibis hierarchy isn't strictly enforced we can add custom
   # entities to the bottom of our menu

   if { $data(-strict) == "no" } {
      $w.popup add separator
      $w.popup add command -label "Custom entry..." -foreground red         \
         -command "ibisIcon_AddCustomType $w"
      set custom [$data(environment) get custom-linktypes]
      foreach i $custom {
         $w.popup add command -label "[lindex $i 1]"                        \
            -command "eval $cmd $w $data(-entity) [lindex $i 0]"
      }
    }
}

# ------------------------------------------------------------------------
# Create a link between this node and its parent
# ------------------------------------------------------------------------

proc ibisIcon_DrawLink { linkstyle w otherW link t } {
   upvar #0 $w ibisIcon
   upvar #0 $otherW otherIcon

   set c1 [$ibisIcon(environment) get coords]
   set c2 [$otherIcon(environment) get coords]
   set small "-*-helvetica-medium-r-*-*-10-*"
   catch { $ibisIcon(-canvas) delete $t }

   #
   # draw in the appropriate symbol onto the line

   if { $link != "none" } {

      #
      # find the mid-point of the line for our appropriate character
      # and draw in the line initially

      set mid [list [expr ([lindex $c2 0] + [lindex $c1 0]) /2]          \
         [expr ([lindex $c2 1] + [lindex $c1 1]) /2]]

      eval "$ibisIcon(-canvas) create line $c2 $mid -fill gray           \
         -arrow first -arrowshape {28 32 8} -tag $t"
      set arrow [eval "$ibisIcon(-canvas) create line $mid $c1           \
         -fill gray -arrow first -arrowshape {28 32 8} -tag $t"]

      if { [string first "custom-" $link] == 0 } {
         set linkstyle text
         set custom [$ibisIcon(environment) get custom-linktypes]
         set idx [cw_ListSearch $custom 0 $link]
         if { $idx != -1 } {
            set link [lindex [lindex $custom $idx] 1]
         } else {
            set link [string range $link 7                               \
               [expr [string length $link] -1]]
         }
      }

      if { $linkstyle == "text" } {
         set txt $link
         switch -exact $link {
            responds    {set txt "responds to"}
            replaces    {set txt "replaces"}
            suggestedby {set txt "is suggested by"}
            objects       {set txt "objects to"}
         }
         eval "$ibisIcon(-canvas) create text $mid -text {$txt}         \
            -font $small -tag $t"

      } else {

         switch -exact $link {
            generalises {
               set chr "\xd9"
               eval "$ibisIcon(-canvas) create text $mid -text $chr      \
                  -font -*-symbol-*-*-*-*-18-* -fill black               \
                  -tag $t"
            }
            specialises {
               set chr "\xda"
               eval "$ibisIcon(-canvas) create text $mid -text $chr    \
                  -font -*-symbol-*-*-*-*-18-* -fill black             \
                  -tag $t"
            }
            responds {
               set chr "\xbf"
               eval "$ibisIcon(-canvas) create text $mid -text $chr    \
                  -font -*-symbol-*-*-*-*-18-* -fill blue              \
                  -tag $t"
            }
            replaces {
               set chr "\xdb"
               eval "$ibisIcon(-canvas) create text $mid -text $chr    \
                  -font -*-symbol-*-*-*-*-18-* -fill red               \
                  -tag $t"
            }
            questions {
               eval "$ibisIcon(-canvas) create text $mid -text {?}     \
                  -font -*-helvetica-bold-r-*-*-24-* -fill goldenrod   \
                  -tag $t"
            }
            suggestedby {
               set chr "\xde"
               eval "$ibisIcon(-canvas) create text $mid -text $chr    \
                  -font -*-symbol-*-*-*-*-18-* -fill blue              \
                  -tag $t"
            }
            supports {
               set chr "\xd6"
               eval "$ibisIcon(-canvas) create text $mid -text $chr    \
                  -font -*-symbol-*-*-*-*-18-* -fill limegreen         \
                  -tag $t"
            }
            objects {
               eval "$ibisIcon(-canvas) create text $mid -text {X}     \
                  -font -*-helvetica-bold-r-*-*-18-* -fill red         \
                  -tag $t"
            }
            custom {
               eval "$ibisIcon(-canvas) create text $mid -text {$link} \
                  -font $small -tag $t"
            }
         }
      }
   }
   $ibisIcon(-canvas) bind $t <Double-Button-2>                        \
                  "ibisIcon_RequestBreakLink $w $otherW"
}

# -------------------------------------------------------------------
# Instead of just breaking the link, fire off an event instead (which
# can be picked up and dealt with accordingly)
# -------------------------------------------------------------------

proc ibisIcon_RequestBreakLink { nodeW parentW } {
   breakLink notify ibisIconBreakLink                                  \
      [list [list N $nodeW] [list P $parentW]]
}
```

```
# ----------------------------------------------------------------------
# Remove a link between a widget and its parent. This routine does NO
# validation checks!!
# ----------------------------------------------------------------------

proc ibisIcon_BreakLink { nodeW } {
   upvar #0 $nodeW ibisIcon
   upvar #0 $ibisIcon(-parent) parentIcon

   set children $parentIcon(-children)
   set item [lsearch $children $nodeW]
   set parentIcon(-children) [lreplace $children $item $item]
   set ibisIcon(-parent) ""
   $ibisIcon(-canvas) delete $ibisIcon(parentLinkTag)
   set ibisIcon(parentLinkTag) ""
}

# ----------------------------------------------------------------------
# Highlight a node with colour "colour" with the option of highlighting
# all its child nodes
# ----------------------------------------------------------------------

proc ibisIcon_Highlight { node colour recurse } {
   upvar #0 $node ibisIcon

   if { ($ibisIcon(origBgColour) == "") && ($colour != "reset") } {
      set ibisIcon(origBgColour) [$node getstate -background]
   }

   if { $colour == "reset" } {
      foreach i $ibisIcon(origBgColour) {
         [lindex $i 0] configure -background [lindex $i 1]
      }
   } else {
      $node setstate -background $colour
   }

   if { $recurse == "all" } {
      foreach i $ibisIcon(attachments) {
         ibisIcon_Highlight [lindex $i 0] $colour no
      }
   }
}

# ----------------------------------------------------------------------
# If the underlying environment changes update the icon views
# ----------------------------------------------------------------------

proc ibisIcon_EnvironmentChanged { w key } {
   upvar #0 $w ibisIcon

   set parts [split $key .]
   set keyItem [lindex $parts 0]
   set lastPart [lindex $parts end]

   if { $keyItem == "coords" } {
      set item [$ibisIcon(environment) get $key]
      set x [lindex $item 0]
      set y [lindex $item 1]

      $ibisIcon(-canvas) coords $ibisIcon(canvasID) $x $y
      set children "$ibisIcon(attachments)"

      foreach child $children {
         if { [lindex $child 3] == "end" } {
            ibisIcon_DrawLink $ibisIcon(-linkstyle) $w                        \
               [lindex $child 0] [lindex $child 1] [lindex $child 2]
         } else {
            ibisIcon_DrawLink $ibisIcon(-linkstyle) [lindex $child 0]         \
               $w [lindex $child 1] [lindex $child 2]
         }
      }
   } elseif { $keyItem == "ibistext" } {
      set item [$ibisIcon(environment) get $key]
      if { [string length $item] > 100 } {
         set item "[string range $item 0 95]..."
      }
      $w.icon configure -text $item
   }
}

# ----------------------------------------------------------------------
# When we receive details of a new environment build up the text and
# choicebutton entities
# ----------------------------------------------------------------------

proc ibisIcon_EnvironmentReceived { w } {
   upvar #0 $w data

   set item [$data(environment) get ibistext]
   if { [string length $item] > 100 } {
      set item "[string range $item 0 95]..."
   }
   $w.icon configure -text $item

   set others [$data(environment) keys choice]
   foreach i $others {
      $w.choices.$i configure                                             \
         -currentvalue [$data(environment) get choice.$i]
   }

   set children [$data(environment) get attachments]
   set data(attachments) $children

   groupIcon_EnvironmentReceived $w
}

# ----------------------------------------------------------------------
# entity popup menu routines
# ----------------------------------------------------------------------

proc ibisIcon_PopupMenu { w x y } {
   $w.popup post $x $y
   grab $w.popup
}

# ----------------------------------------------------------------------
# By default, the new entitiy commands do nothing
# ----------------------------------------------------------------------

proc ibisIcon_Menu { w entity link } {
   cw_Debug "ibisIcon_Menu: no command override" red
}

# ----------------------------------------------------------------------
# Add or delete an attachment
# ----------------------------------------------------------------------

proc ibisIcon_ChangeAttachments { w attachments } {
   upvar #0 $w data
   set orig $data(attachments)
   foreach i $attachments {
      set node [lindex $i 0]
      set link [lindex $i 1]

      set idx [cw_ListSearch $orig 0 $node]
      if { $idx == -1 } {
         #
         # the attachment is new, so let's add it

         set linktag "link[cw_GetCounter]"
         lappend data(attachments) [lappend i $linktag end]

         #
         # as there's no parent node, we also add the attachment to
         # the other nodes' attachment details

         upvar #0 $node otherData
         lappend otherData(attachments) [list $w $link $linktag start]

         ibisIcon_DrawLink $data(-linkstyle) $w $node $link $linktag
         $data(environment) set attachments $data(attachments)
         $otherData(environment) set attachments $otherData(attachments)

      } elseif { [lindex [lindex $i $idx] 1] != $link } {
         #
         # the type of link between these entities has changed

         puts "ENTITY LINK CHANGED"
      }
   }
}

# ----------------------------------------------------------------------
# return TRUE if we can attach one entity type to another
# ----------------------------------------------------------------------

proc ibisIcon_CanAttach { w child } {
   upvar #0 $w data
   upvar #0 $child childdata

   if { $w == $child } { return 0 }

   if { $data(-strict) == "no" } { return 1 }

   set cdata $childdata(-entity)
   set e $data(-entity)

   if { ($e == "issue") && ($cdata == "argument") } { return 0
   } elseif {($e == "argument") && ($cdata != "issue")} { return 0
   } elseif {($e == "position") && ($cdata == "position")} {return 0}

   return 1
}

# ----------------------------------------------------------------------
# Popup a dialog for changing the issue text
# ----------------------------------------------------------------------

proc ibisIcon_NodeDialog { w } {
   upvar #0 $w data
   set d ".ibis[cw_ChangeDotInName $w]"
   if [winfo exists $d] { raise $d; return }
   set e "g$d"
   global $e
   set $e $data(-entity)

   toplevel $d
   wm title $d "IBIS node attributes"
   frame $d.f -bd 2 -relief groove
   radiobutton $d.f.r1 -text "Issue" -variable $e -value issue
   radiobutton $d.f.r2 -text "Position" -variable $e -value position
   radiobutton $d.f.r3 -text "Argument" -variable $e -value argument

   if { $data(attachments) != {} } {
      $d.f.r1 configure -state disabled
      $d.f.r2 configure -state disabled
      $d.f.r3 configure -state disabled
   }

   text $d.text -width 20 -height 3
   $d.text insert 1.0 [$w.icon cget -text]

   frame $d.f2
   button $d.f2.cancel -text "Cancel" -command "destroy $d"
   button $d.f2.ok -text "Apply" -command "ibisIcon_Apply $w $d $e"

   pack $d.f -fill x
   pack $d.f.r1 $d.f.r2 $d.f.r3 -side left -fill x -expand yes
   pack $d.text -fill both -expand yes
   pack $d.f2
   pack $d.f2.cancel $d.f2.ok -side left -padx 1c -fill x -expand yes
}

# ----------------------------------------------------------------------
# Apply the changes to the dialog
# ----------------------------------------------------------------------

proc ibisIcon_Apply { w d e } {
   upvar #0 $w data
   upvar #0 $e entity

   if { $entity != $data(-entity) } {
      set data(-entity) $entity
      ibisIcon_BuildPopup $w $data(-menucmd)
   }
   set txt [$d.text get 1.0 end]
   set txt [string trim $txt "\n"]
   $w configure -text $txt
   destroy $d
}

# ----------------------------------------------------------------------
# Called whenever we click on a choicebutton
# ----------------------------------------------------------------------

proc ibisIcon_ChoiceClick { b val } {
   set w [file rootname [file rootname $b]]
   upvar #0 $w data

   gk_viewsSetAttribute $data(-scrollid) $data(local) choice $val
   $data(environment) set choice.$data(local) $val
}

# ----------------------------------------------------------------------
# Dialog for creating a new custom type link
# ----------------------------------------------------------------------
```

```
proc ibisIcon_AddCustomType { w } {
   set d ".ibismenu[cw_ChangeDotInName $w]"
   if [winfo exists $d] { raise $d; return }

   toplevel $d
   wm title $d "IBIS custom link"
   label $d.l -text "Custom link text:"
   entry $d.e
   button $d.close -text "Cancel" -command "destroy $d"
   button $d.add -text "Add" -command "ibisIcon_AddCustomTypeOK $w $d"

   pack $d.l $d.e -side left
   pack $d.close -side bottom
   pack $d.add -side left -padx 5m
}

# -----------------------------------------------------------------------
# Actually create the new custom link type
# -----------------------------------------------------------------------

proc ibisIcon_AddCustomTypeOK { w d } {
   upvar #0 $w data

   set name [$d.e get]
   set linkname "custom-[cw_RemoveSpacesFromName $name 1]"
   set custom [$data(environment) get custom-linktypes]
   if { [cw_ListSearch $custom 0 $linkname] == -1 } {
      $w addcustom $linkname $name
      ibisIcon_BuildPopup $w $data(-menucmd)
      eval "$data(-menucmd) $w $data(-entity) $linkname"
      destroy $d
   } else {

      #
      # the link already exists

      puts "\a"
   }
}
```

E.7 Choicebutton utility widget

```
# -----------------------------------------------------------------------
# choicebutton.tcl
# TCL override of the checkbutton widget
#
# developed by Keith McAlpine
#
# -----------------------------------------------------------------------

# -----------------------------------------------------------------------
# Basic design philosophy:
#
#   Used to allow multiple rotating options for a checkbutton. The
#   default setting is undecided (?) yes or no
#
# Configuration options:
#   -values          - a list of possible values (names must be bitmaps)
#   -currentvalue    - the current value of the widget
#   -callcmd         - a cmd to execute whenever the widget changes state
# -----------------------------------------------------------------------

set d "/phd/gk/cw/bitmaps"
```

```
# ---------------------------------------------------------------------
# Routines used by the ClassBuilder routines in Groupkit for the easy
# creation of widgets
# ---------------------------------------------------------------------

proc choicebutton {w args} {
   # Create a choicebutton
   eval gkInt_CreateWidget $w choicebutton choicebuttonClass $args
   return $w
}

proc choicebutton_CreateClassRec {} {
   global choicebutton d

   # Setup the class record for the choicebutton.
   # It inherits all the commands/options from the canvas

   set choicebutton(inherit)    {button}
   set choicebutton(options)    {-values -currentvalue -callcmd}
   set choicebutton(methods)    {}

   set values [list undecided yes no]
   set choicebutton(-values)    [list -values values Values $values]
   set choicebutton(-callcmd) {-callcmd callCmd CallCmd none}
   set choicebutton(-currentvalue)                                   \
      {-currentvalue currentValue CurrentValue undecided}
}

proc choicebutton_ConstructWidget { w } {
   upvar #0 $w data
   global d
   button $w.b -bitmap "@$d//$data(-currentvalue)"                   \
      -command "choicebutton_Press $w"
   pack $w.b
}

# ---------------------------------------------------------------------
# Routines used internally by the choicebutton widget
# ---------------------------------------------------------------------

proc choicebutton_InitWidgetRec {w class className args} {
   upvar #0 $w data
}

# ---------------------------------------------------------------------
# Routine called on the result of a <widget> configure xxx command
# ---------------------------------------------------------------------

proc choicebutton_Config {w option args} {
   upvar #0 $w data
   global d
   set args [lindex $args 0]

   switch -exact [string range $option 1 end] {
      currentvalue    {
         set data(-currentvalue) $args
         $w.b configure -bitmap "@$d//$data(-currentvalue)"
      }
      values          {}
      callcmd         {}
      default {
         $w.b configure $option $args
      }
   }
}
```

```
# ------------------------------------------------------------------------
# Routine called on result of a <widget> <command> xxx command
# ------------------------------------------------------------------------

proc choicebutton_Methods {w command args} {
   upvar #0 $w data

   set args [lindex $args 0]

   switch -exact $command {
      default    {
         eval "$w.b $command $args"
      }
   }
}

# ------------------------------------------------------------------------
# Method to update the button icon and call a callig routine if we
# want some feedback of the current state
# ------------------------------------------------------------------------

proc choicebutton_Press { w } {
   upvar #0 $w data
   global d

   set idx [lsearch $data(-values) $data(-currentvalue)]
   incr idx
   if { $idx >= [llength $data(-values)] } { set idx 0 }

   set data(-currentvalue) [lindex $data(-values) $idx]
   $w.b configure -bitmap "@$d//$data(-currentvalue)"

   if { $data(-callcmd) != "none" } {
      eval $data(-callcmd) $w $data(-currentvalue)
   }
}
```

E.8 Multitext groupware text entry widget

```
# ------------------------------------------------------------------------
# Multitext widget
#
# Based on the gkText widget but with some major enhancements -
# revision control & locking
#
#
# Data structures:
# The multitext widget is based on the groupicon widget. Thus it has
# coupling features as in groupicon along with an underlying
# 'environment' data structure. The shared data for the multitext is
# contained inside this environment structure. Localised data for the
# multitext is contained in the localised data(xyz) widget array.
#
# The multitext widget uses the concept of 'editing sessions' for
# revision management. A new editing session is created whenever a
# person enters the widget, or whenever text BY ANOTHER AUTHOR is
# changed. A single author can simulate multiple sessions by exiting
# and reentering the widget.
# The data is stored as a series of deltas from the basic document
# (EMPTY) baseline.
#
#
```

```
# ENVIRONMENT STRUCTURE:
#   env.curr_session           - the current session number (latest value)
#   env.text.<session num>     - the set of text deltas for the current
#                                session. These deltas consist of a list of
#                                elements made up of:
#          * <username of who changed text>
#          * <insertion | deletion>
#          * <insertion point of the change>
#          * <the text changed> (i.e. the inserted or the deleted text)
#
#   env.locks.<username>     - list of locks by particular users.Locks are
#                              long-lived by default but may be overridden
#                              on exit from the node or after a specific
#                              time (decided by a config option)
#
# COMMANDS:
#   <widget> initialise    - initialise the widget (only user present) by
#                            first seeing if there's a file and then
#                            setting default values if not.
#
# CONFIGURATIONS:
#   <widget> -locktime    - time before lock expires
#                           (infinite | onexit | xxx (hour|min|second)
#   <widget> -showcolour   - 1 | 0 for whether the widget entries are
#                            displayed in the author colours or in black
#                            with grey for deletions.
#
# -----------------------------------------------------------------------

# IMPORTANT NOTE
# This widget assumes there's a routine to determine the colour from a
# username defined elsewhere.
# It also assumes each user has a cursor_color attribute defined for
# the colour of their cursor (generally a lighter vsn of their color)

# This code originates from the gkText widget code in GroupKit, esp.
# with regard to the binding sets and selection mechanism.
#
# It also includes code for changing a default colour by switching
# from RGB to HSB colour formats - this code is taken from:
# page 616 of "Fundamentals of Interactive Computer Graphics"
# by Foley and Van Dam.

# -----------------------------------------------------------------------
# Routines used by the ClassBuilder routines in Groupkit for the easy
# creation of widgets
# -----------------------------------------------------------------------

proc multitext {w args} {

   #
   # Create a multitext widget

   eval gkInt_CreateWidget $w multitext multitextClass $args
   return $w
}

proc multitext_CreateClassRec {} {
   global multitext

   groupIcon_CreateClassRec

   #
   # Setup the class record for the grouptable.
   # It inherits all the commands/options from the canvas
   set multitext(inherit) {text}
   set multitext(options) {-envname -widget -icontext -coupling         \
                           -canvas -defaulticon -scrollid -locktime      \
                           -showcolour }
   set multitext(methods) { adduser attributechanged deleteuser          \
                           moveuser initialise save load }

   set multitext(-envname)      {-envname envName EnvName ""}
   set multitext(-widget)       {-widget widget Widget error}
   set multitext(-icontext)     {-icontext iconText IconText <icon>}
   set multitext(-coupling)     {-coupling coupling Coupling forcenone}
   set multitext(-canvas)       {-canvas canvas Canvas .}
   set multitext(-defaulticon)  {-defaulticon defaultIcon DefaultIcon ""}
   set multitext(-scrollid)     {-scrollid scrollId ScrollId none}
   set multitext(-locktime)     {-locktime lockTime LockTime infinite}
   set multitext(-showcolour)   {-showcolour showColour ShowColour 1}
}

proc multitext_ConstructWidget { w } {
   upvar #0 $w data

     text $data(widget).t -exportselection false
   pack $data(widget).t -fill both -expand yes

   bindtags $data(widget).t [list . all $data(widget).t gkTextTag]
   bind gkTextTag <Enter> "gkTextBind $data(widget).t Enter"
   bind gkTextTag <FocusIn> "gkTextBind $data(widget).t FocusIn"

   #
   # set up the bindings for middle-button presses over tags

   $data(widget).t tag bind sel_[users local.usernum] <2>              \
      "multitext_SelectionPopup $w [users local.usernum] %X %Y"
   $data(widget).t tag bind sel_[users local.usernum]                  \
      <ButtonRelease-2> [list catch "destroy $data(widget).t.popup"]
}

# -----------------------------------------------------------------------
# Routines used internally by the multitext widget
# -----------------------------------------------------------------------

proc multitext_InitWidgetRec {w class className args} {
   upvar #0 $w data

   groupIcon_InitWidgetRec $w $class $className $args

   # Initialize all the variables needed for the multitext

   set data(widget) $w
   set data(usersInWidget) {}
   set data(local) [users local.usernum]
   set data(colour) [userprefs color]
   set data(username) [cw_RemoveSpacesFromName [users local.username] 0]

   #
   # for lighter and darker shades, we need to create a temp button for
   # making the changes. ??? There must be a better way, but I don't
   # want to switch to 'C' !!!

   catch { button .colourset }

   .colourset configure -bg $data(colour)
   set rgb [winfo rgb .colourset $data(colour)]

   set red [lindex $rgb 0]
   set green [lindex $rgb 1]
```

```
    set blue [lindex $rgb 2]
    set hsb [rgbToHsv $red $green $blue]

    set hue [lindex $hsb 0]
    set sat [lindex $hsb 1]
    set bri [lindex $hsb 2]

    set b [expr $bri * 0.6]
    set rgb [hsbToRgb $hue $sat $b]
    set data(delete_colour) [eval "format \"#%04x%04x%04x\" $rgb"]
    if { $bri > 0.75 } {
       set s [expr $sat *0.3]
       set b $bri
    } else {
       set s $sat
       set b [expr $bri * 1.75]
    }
    set rgb [hsbToRgb $hue $s $b]
    set data(lock_colour) [eval "format \"#%04x%04x%04x\" $rgb"]
    if { $bri > 0.75 } {
       set s [expr $sat *0.1]
       set b $bri
    } else {
       set s $sat
       set b [expr $bri * 2.5]
    }
    set rgb [hsbToRgb $hue $s $b]
    set data(select_colour) [eval "format \"#%04x%04x%04x\" $rgb"]
    set data(orig_colour) $data(colour)
    set data(orig_delete_colour) $data(delete_colour)
    set data(orig_lock_colour) $data(lock_colour)
    set data(orig_select_colour) $data(select_colour)

    destroy .colourset
}

proc multitext_DestroyWidget { w } {
    groupIcon_DestroyWidget $w
}

# -----------------------------------------------------------------------
# Convert an RGB value into an HSB list
# -----------------------------------------------------------------------

proc rgbToHsv {red green blue} {
     if {$red > $green} {
    set max $red.0
    set min $green.0
     } else {
    set max $green.0
    set min $red.0
     }
     if {$blue > $max} {
    set max $blue.0
     } else {
    if {$blue < $min} {
        set min $blue.0
    }
     }
     set range [expr $max-$min]
     if {$max == 0} {
```

```
    set sat 0
     } else {
    set sat [expr {($max-$min)/$max}]
     }
     if {$sat == 0} {
    set hue 0
     } else {
    set rc [expr {($max - $red)/$range}]
    set gc [expr {($max - $green)/$range}]
    set bc [expr {($max - $blue)/$range}]
    if {$red == $max} {
        set hue [expr {.166667*($bc - $gc)}]
    } else {
        if {$green == $max} {
      set hue [expr {.166667*(2 + $rc - $bc)}]
        } else {
      set hue [expr {.166667*(4 + $gc - $rc)}]
        }
    }
    if {$hue < 0.0} {
        set hue [expr $hue + 1.0]
    }
     }
     return [list $hue $sat [expr {$max/65535}]]
}

# ----------------------------------------------------------------------
# Convert HSB values into an RGB list
# ----------------------------------------------------------------------

proc hsbToRgb {hue sat value} {
     set v [format %.0f [expr 65535.0*$value]]
     if {$sat == 0} {
    return "$v $v $v"
     } else {
    set hue [expr $hue*6.0]
    if {$hue >= 6.0} {
        set hue 0.0
    }
    scan $hue. %d i
    set f [expr $hue-$i]
    set p [format %.0f [expr {65535.0*$value*(1 - $sat)}]]
    set q [format %.0f [expr {65535.0*$value*(1 - ($sat*$f))}]]
    set t [format %.0f [expr {65535.0*$value*(1 - ($sat*(1 - $f)))}]]
    case $i \
        0 {return "$v $t $p"} \
        1 {return "$q $v $p"} \
        2 {return "$p $v $t"} \
        3 {return "$p $q $v"} \
        4 {return "$t $p $v"} \
        5 {return "$v $p $q"}
    error "i value $i is out of range"
     }
}

# ----------------------------------------------------------------------
# Routine called on the result of a <widget> configure xxx command
# ----------------------------------------------------------------------

proc multitext_Config {w option args} {
    upvar #0 $w data
    set args [lindex $args 0]
    switch -exact [string range $option 1 end] {
      scrollid {
         set data(-scrollid) $args
         gk_views $w $data(-scrollid)
         gk_viewsSetCoords $data(-scrollid) $data(local) [list 0 0 1 1]
         gk_viewsSetAttribute $data(-scrollid) $data(local)             \
                              color [userprefs color]
      }

      envname {
         groupIcon_Config $w $option $args
      }

      locktime {
      }

      showcolour {
         multitext_ShowWidgetColour $w $args
      }

      coupling { groupIcon_Config $w $option $args }

      default { $data(widget).t config $option $args }
   }
}


# ----------------------------------------------------------------------
# Routine called on result of a <widget> <command> xxx command
# ----------------------------------------------------------------------

proc multitext_Methods {w command args} {
   upvar #0 $w data
   set args [lindex $args 0]

   cw_Debug "multitext method: $command - $args - $w" white

   switch -exact $command {
      adduser          { multitext_UpdateUserList $w }
      deleteuser       { multitext_UpdateUserList $w }

      attributechanged {
          if { [lindex $args 2] == "color" } {}
      }

      initialise      { multitext_Initialise $w }
      save            { multitext_SaveData $w }
      load            { multitext_LoadData $w }
      moveuser        {}
      setstate        { groupIcon_Methods $w $command $args }
      getstate        { groupIcon_Methods $w $command $args }
      getenvironment  { groupIcon_Methods $w $command $args }
      bindall         { groupIcon_Methods $w $command $args }
      getcanvasinfo   { groupIcon_Methods $w $command $args }

      default   {
         puts "doing default option in multitext: $command"
         eval "$data(widget).t $command $args"
      }
   }
}

# ----------------------------------------------------------------------
# Build up the menu for popuping up when you have a selection
# ----------------------------------------------------------------------
```

```
proc multitext_SelectionPopup { w usr x y } {
   upvar #0 $w data

   catch {destroy $data(widget).t.popup}
   set m "$data(widget).t.popup"
   menu $m -tearoff 0
   $m add command -label "Lock"                                              \
      -command [list multitext_Lock $w $data(widget).t $usr]
   tk_popup $m $x $y
}

# ----------------------------------------------------------------------------
# Perform a text lock to prevent others in the widget. There's no
# checks yet for whether something has already been locked ????
# ----------------------------------------------------------------------------

proc multitext_Lock { W w usr } {
   upvar #0 $W data
   multitext_DoLock $w $usr $data(lock_colour)
   multitext_TypingTo multitext_DoLock $w $usr $data(lock_colour)
}

proc multitext_DoLock { w usr colour } {
   eval "$w tag add lock_$usr [$w tag ranges sel_$usr]"
   $w tag configure lock_$usr -background $colour
}

# ----------------------------------------------------------------------------
# Read in any disk data (if there is any) - otherwise basic initialise
# ----------------------------------------------------------------------------

proc multitext_Initialise { w } {
   upvar #0 $w data

   set env $data(environment)

   #
   # first see if there is a file on our local filing system

   if [file exists "$env.cwnode"] {
      multitext_LoadData $w
      $env set curr_session [expr [$env get curr_session] + 1]
      $env set curr_tag 0
   } else {
      set fileID [open "$env.cwnode" w+]
      close $fileID

      #
      # initialise our environment data and structures

      $env set curr_session 1
      $env set curr_tag 0
      multitext_SetCursor $w.t 1.0 left
   }
}

# ----------------------------------------------------------------------------
# Send our current details over to a new entrant
# As this is done via the widget data and not the environment,
# syncronisation problems could arise. It would be better to put all the
# changes into the environment first and just send it.
#
# For now, we just send the widget details, filling in the environment
# on final exit.
# ----------------------------------------------------------------------------
```

```
proc multitext_SendDetailsTo { w newuser } {
   upvar #0 $w data
   set env $data(environment)

   gk_toUserNum $newuser $env set curr_session [$env get curr_session]
   gk_toUserNum $newuser $env set curr_tag [$env get curr_tag]

   set txt [lsort -command multitext_SortTheTags [$env keys data]]

   foreach i $txt {
     gk_toUserNum $newuser $env set data.$i.pos [$env get data.$i.pos]
     gk_toUserNum $newuser $env set data.$i.text [$env get data.$i.text]
   }

   #
   # now copy over our text widget elements.

   foreach i $txt {
      set pos [$w.t index $i]
      set strings [$w.t tag ranges $i]
      set parts [split $i "-"]
      set thistext ""

      for {set idx 0} {$idx < [llength $strings] } {incr idx 2} {
         set thistext "$thistext[$w.t get                                \
            [lindex $strings $idx] [lindex $strings [expr $idx + 1]]]"
      }

      if { [lindex $parts 2] == "INS" } {
          gk_toUserNum $newuser $w.t insert $pos $thistext $i
         set ul [$w.t tag cget $i -underline]
         gk_toUserNum $newuser $w.t tag configure $i -underline $ul
      } else {

         #
         # the text here is a deletion - it's already been added so just
         # strike it out - we'll hide it later though if its from an
         # early session!

         gk_toUserNum $newuser $w.t tag add $i $pos                    \
            [$w.t index "$pos + [string length $thistext] chars"]
         set ov [$w.t tag cget $i -overstrike]
         gk_toUserNum $newuser $w.t tag configure $i -overstrike $ov
      }
      set fg [GetColourForUserName [lindex $parts 3]]
      gk_toUserNum $newuser multitext_SetTagColour $w.t $i $fg
   }

   #
   # now find all the marks we've got placed in the widget and insert
   # them into the new one

   set marks [$w.t mark names]
   foreach m $marks {
      set idx [$w.t index $m]
      gk_toUserNum $newuser $w.t mark set $m $idx
      # gk_toUserNum $newuser $w.t mark gravity $m left
   }

   gk_toUserNum $newuser multitext_SetCursor $w.t 1.0 left
}

# ------------------------------------------------------------------
# To save the data, first we put the info in the environment so that
# there are no refs to the widget information - then we dump the
# environment to disk (file: <envname>.cwnode)
# ------------------------------------------------------------------
```

```
proc multitext_SaveData { w } {
   upvar #0 $w data

   set env $data(environment)
   set txt [lsort -command multitext_SortTheTags [$env keys data]]

   foreach i $txt {
      $env set data.$i.pos [$w.t index $i]

      set strings [$w.t tag ranges $i]
      set thistext ""

      for {set idx 0} {$idx < [llength $strings] } {incr idx 2} {
         set thistext "$thistext[$w.t get                                    \
            [lindex $strings $idx] [lindex $strings [expr $idx + 1]]]"
      }

      $env set data.$i.text $thistext
   }

   #
   # now dump the environment to disk

   set fileID [open "$env.cwnode" w+]
   set keys [$env keys]
   foreach i $keys {
      puts $fileID $i
      puts $fileID [$env get $i]
   }
   close $fileID
}

# -----------------------------------------------------------------------
# Load in the environment data from a file and build the widget
# afterwards.
# -----------------------------------------------------------------------

proc multitext_LoadData { w } {
   upvar #0 $w data

   set env $data(environment)

   #
   # first read the environment from disk

   if { ![file exists "$env.cwnode"] } { return }
   set fileID [open "$env.cwnode" r]
   while { ![eof $fileID] } {
      gets $fileID envName
      gets $fileID envData

      #
      # the environment could be stored over more than 1 line
      # (eg. where the text has returns), so parse it for numbers of {}

      set openbrackets [ cw_NumberOf $envData "\{" ]
      set closebrackets [ cw_NumberOf $envData "\}" ]

      while { $openbrackets > $closebrackets } {
         gets $fileID nextline
         incr openbrackets [ cw_NumberOf $nextline "\{" ]
         incr closebrackets [ cw_NumberOf $nextline "\}" ]
         append envData "\n$nextline"
      }

      if { $openbrackets == 0 } {
         $env set $envName $envData
      } else {
         $env import $envName $envData
      }
   }
   close $fileID

   #
   # now we need to build up our widget text, going from the first
   # session to the last and performing each tag action in turn

   set taghist [lsort -command multitext_SortTheTags [$env keys data]]

   foreach i $taghist {
      set parts [split $i "-"]
      set pos [$env get data.$i.pos]
      $w.t mark set $i $pos
      $w.t mark gravity $i left

      if { [lindex $parts 2] == "INS" } {
         $w.t insert $pos "[$env get data.$i.text]" $i
      } else {

         #
         # do the delete (we should make the deletion darker)

         $w.t tag add $i $pos [$w.t index                              \
            "$pos + [string length "[$env get data.$i.text]"] chars"]
         $w.t tag configure $i -overstrike 1
      }

      # NON-PORTABLE !!! - we call a utility toutine to convert the
      # username into a colour, a routine which is only useful from
      # the editor tool (where it's created)

      set fg [GetColourForUserName [lindex $parts 3]]
      multitext_SetTagColour $w.t $i $fg
   }
}

# -----------------------------------------------------------------------
# Sort routine used to order the tags created by session/creation style
# -----------------------------------------------------------------------

proc multitext_SortTheTags { arg1 arg2 } {
   set parts1 [split $arg1 "-"]
   set parts2 [split $arg2 "-"]

   if { [lindex $parts1 0] < [lindex $parts2 0] } { return -1 }
   if { [lindex $parts1 0] > [lindex $parts2 0] } { return 1 }

   if { [lindex $parts1 1] < [lindex $parts2 1] } { return -1 }
   if { [lindex $parts1 1] > [lindex $parts2 1] } { return 1 }

   cw_Debug "******* TWO MULTITEXT TAGS ARE THE SAME !!! ******" red
   return 0
}

# -----------------------------------------------------------------------
# Routine to either switch off or show all the author colours used in
# the widget contents. Colours are worked out as in the LoadData routine
# for showing, or black & grey if not showing.
# -----------------------------------------------------------------------

proc multitext_ShowWidgetColour { w newstate } {
   upvar #0 $w data
   set env $data(environment)
   set tags [lsort -command multitext_SortTheTags [$env keys data]]

   foreach i $tags {
      set p [split $i "-"]
      multitext_SetTagColour $w.t $i                                        \
                               [GetColourForUserName [lindex $p 3]]
   }
}

# ----------------------------------------------------------------------
# Set a tags colour to be either the colour wanted, or black/grey
# ----------------------------------------------------------------------

proc multitext_SetTagColour { w tag usercolour } {
   set w [file rootname $w]
   upvar #0 $w data

   if { [$w cget -showcolour] == 0 } {
      set parts [split $tag "-"]
      if { [lindex $parts 2] == "INS" } { set usercolour black
      } else { set usercolour grey }
   }

   $w.t tag configure $tag -foreground $usercolour
}

# ----------------------------------------------------------------------
# Routines to send to the groupIcon - need to do this because the
# Classbuilder doesn't have a multiple inheritance mechanism
# ----------------------------------------------------------------------

proc multitext_EnvironmentReceived { w } {
   groupIcon_EnvironmentReceived $w
}

proc multitext_EnvironmentChanged { w key } {
   groupIcon_EnvironmentChanged $w
}

proc multitext_GetParent { w } {
   return [winfo parent $w]
}

# ----------------------------------------------------------------------
# Make a tag made up of:
#   <session num>-<running tag num>-<tag type (INS | DEL)>-<username>
# ----------------------------------------------------------------------

proc multitext_MakeTag { w type } {
   global gNewTag

   upvar #0 $w data
   set gNewTag 1
   set env $data(environment)
   set tagnum [expr [$env get curr_tag] + 1]
   $env set curr_tag $tagnum

   return "[$env get curr_session]-$tagnum-$type-$data(username)"
}

# ----------------------------------------------------------------------
# Routine to find the tag which we are inside, and adding to if
# necessary. We look to the left to see what tag we are building onto.
# A side effect of this is that entering new text at the beginning of a
# block will always appear as a new tag insertion.
# ---------------------------------------------------------------------

proc multitext_GetContainingTag { w pos type } {
   global gNewTag

   set W [file rootname $w]
   upvar #0 $W data

   set gNewTag 0

   #
   # we look to the LEFT of the insertion point to see what tag there
   # is if it's an insertion or a deletion. If its a deletion, we also
   # look to the RIGHT to see if there's already a deletion which we
   # can just append onto

   set tags [cw_ReverseList                                              \
             [$w tag names [$w index "insert_$data(local) - 1c"]]]
   set deltag ""

   if { $type == "DEL" } {
      set deltags [cw_ReverseList [$w tag names "insert_$data(local)"]]
      set searchfor                                                      \
         "[$data(environment) get curr_session]-*-$type-$data(username)"

      foreach tag $deltags {
         if [string match $searchfor $tag] { set deltag $tag; break }
      }
   }

   if { $tags == "" } {

      #
      # this is a newly created widget with no tags defined yet. Thus
      # create the tag and assign it to the text entered.

      return [multitext_MakeTag $W $type]
   } else {

      set foundtag DOUBLEDELETE
      foreach tag $tags {
         set parts [split $tag "-"]
         set tagtype [lindex $parts 2]

         if { ($type == "DEL") && [regexp "insert_" $tag] } {

            #
            # we're trying to ins or del past a marker, so keep it as
            # DOUBLEDEL

            return $foundtag
         }

         if { ($tagtype == "INS") || ($tagtype == "DEL") } {

            #
            # we've found the containing tag - but is it our orig one?

            if { ($type == "DEL") && ($tagtype == "DEL") } {

               #
               # when we're deleting into a deletion, keep the tag as
               # DOUBLEDELETE and let the calling proc decide what to do
            } elseif { [lindex $parts 3] == $data(username) } {
                #
                # it's one we created, but is it from this session ?
                if { [lindex $parts 0] == [$data(environment) get         \
                                            curr_session] } {
                  #
                  # the tag is from this session, so return it
                  set foundtag $tag
                } else {
                  #
                  # no it's from an earlier session,so create a new tag
                  set foundtag [multitext_MakeTag $W $type]
                }
            } else {
                set foundtag [multitext_MakeTag $W $type]
            }
            if { $gNewTag && ($type == "DEL") } {
               #
               # if we're doing a deletion, check on the tag to the
               # right to see if it's the same type but keep gNewTag
               # flag as TRUE
               if { $deltag != "" } { set foundtag $deltag }
            }
            return $foundtag
         }
      }
   }
   puts "\7 ---------- INVALID TAG for $w, $pos, $type ----------"
   #
   # if we've got an invalid tag (eg. we're at a marker) then create a
   # new tag and return it (if we're inserting)
   if { $type == "INS" } {
      return [multitext_MakeTag $W $type]
   }
   puts "INVALID TAG returned"
   return "<INVALID TAG>"
}
# ---------------------------------------------------------------------------
# Routine to update the list of users in the view
# ---------------------------------------------------------------------------
proc multitext_UpdateUserList { w } {
   upvar #0 $w data
   set data(usersInWidget) [gk_viewsUsers $data(-scrollid)]
}
# ---------------------------------------------------------------------------
# Routine to serialize the typing calls through a unique user, firing
# messages only to confs which we're interested in (apart from ourself!)
# ---------------------------------------------------------------------------
proc multitext_TypingTo { args } {
   set w [winfo parent [lindex $args 1]]
```

```
    upvar #0 $w data
    set users $data(usersInWidget)

    set idx [lsearch $users $data(local)]
    if { $idx != -1 } { set users [lreplace $users $idx $idx] }
    if { [llength $users] > 0 } {
       eval gk_toUserNum [_gk_getUniqueUser]                              \
                                     gk_toUsers [list $users] $args
    }
}

# -----------------------------------------------------------------------
# Routine to move the cursor position back depending on how many user
# cursors there are to the left of the line - only the originator calls
# this (eg. whoever inserts a char or moves their cursor) and the
# recipients move the cursor to the right if need be (if they have other
# cursors showing as well)
# -----------------------------------------------------------------------

proc multitext_AdaptPosForCursors { w pos usernum operation } {
    set numCursors 0
    set parts [split $pos "."]
    set thisline [lindex $parts 0]

    set tags [$w tag names]
    set numtags [llength $tags]
    set idx 0
    for { set idx 0 } { $idx < $numtags } {incr idx } {
       set item [lindex $tags $idx]
       if { [regexp "insert_*" $item] } {
          if { ![regexp $usernum $item] } {
             set rhs [lindex [$w tag ranges $item] 1]
             if { [lindex [split $rhs "."] 0] == $thisline } {
                if [$w compare $rhs < $pos] {
                   puts "$operation $item"
                   incr numCursors
                }
             }
          }
       }
    }

    set charpos [expr [lindex $parts 1] $operation $numCursors]
    if { $charpos < 0 } { set charpos 0 }
    return "$thisline.$charpos"
}

# -----------------------------------------------------------------------
# Deletion routine - intelligently skips remote insertion points, etc.
# -----------------------------------------------------------------------

proc multitext_TextDelete { w } {
    global gNewTag

    set W [file rootname $w]
    upvar #0 $W data

    set usernum $data(local)
    set env $data(environment)

    set lock [multitext_CheckForLock $W $w]
    if { $lock == "locked" } { return }

    set thistag [multitext_GetContainingTag $w insert_$usernum DEL]
```

```
   #
   # make a note of the start and finish indices of the deletion

   set startpos [$w index "insert_$usernum - 1 char"]
   set endpos [$w index insert_$usernum]
   set insertion 1.0

   #
   # if we're deleting a deletion, just move the cursor

   if { $thistag == "DOUBLEDELETE" } {
      multitext_SetCursor $w $startpos left
      return
   }

   if { $gNewTag } {

      #
      # we're deleting text which was created by someone else, so use
      # strikeout and mark is as deleted. Tk will sort out any splits
      # of other blocks of text as a result.

      $env set data.$thistag.pos MARK
      $env set data.$thistag.text DEFINED-IN-WIDGET
      $w mark set $thistag $startpos
      $w mark gravity $thistag left
      $w tag add $thistag $startpos $endpos
      $w tag configure "$thistag" -overstrike 1
      multitext_SetTagColour $w $thistag $data(delete_colour)
      set insertion $startpos

      set endpos [multitext_AdaptPosForCursors $w $endpos              \
                                            [users local.usernum] "-"]
      set startpos [$w index "$endpos - 1 char"]
      multitext_TypingTo                                                \
         gkTextDoDelete $w $usernum $startpos $endpos $thistag
   } else {

      #
      # we're deleting text which we've added in this session - so no
      # overstrike is performed and no revision mgmt is used. Text
      # typed in this session and deleted by the SAME USER is regarded
      # as simple corrections of no importance.

      $w delete $startpos $endpos

      set endpos [multitext_AdaptPosForCursors $w $endpos              \
                                            [users local.usernum] "-"]
      set startpos [$w index "$endpos - 1 char"]

      multitext_TypingTo gkTextDoDelete $w $usernum $startpos $endpos 0
      set insertion [$w index insert]
   }

   $w see insert
   multitext_SetCursor $w $insertion left
}

# ----------------------------------------------------------------------
# Actually do the text deletion, adapting as need be the cursors
# ----------------------------------------------------------------------

proc gkTextDoDelete { w usernum startpos endpos deltag } {
   set W [file rootname $w]
   upvar #0 $W data
```

```
    set startpos [multitext_AdaptPosForCursors $w $startpos                  \
                                      [users local.usernum] "+"]
    set endpos [multitext_AdaptPosForCursors $w $endpos                      \
                                      [users local.usernum] "+"]

    if { $deltag != 0 } {
       $w mark set $deltag $startpos
       $w mark gravity $deltag left
       $w tag add $deltag $startpos $endpos
       $w tag configure "$deltag" -overstrike 1
       multitext_SetTagColour $w $deltag $data(delete_colour)
    } else {
       $w delete $startpos $endpos
    }
}

# ----------------------------------------------------------------------
# See if the position we are at contains a lock, and return either
# nolock, mylock or locked
# ----------------------------------------------------------------------

proc multitext_CheckForLock { w widget } {
    upvar #0 $w data

    set usernum $data(local)

    set tags [$widget tag names insert_$usernum]
    set idx [lsearch $tags "lock_*"]

    if { $idx != -1 } {

       #
       # there is a lock in place here, so see if its our lock

       if { [lindex $tags $idx] != "lock_$usernum" } {
          puts "\a Locked!"
          return "locked"
       } else {
          return "mylock"
       }
    }

    return "nolock"
}

# ----------------------------------------------------------------------
# multitext_TextInsert --
# Insert a string into a text at the point of the insertion cursor.
# If there is a selection in the text, and it covers the point of the
# insertion cursor, then delete the selection before inserting.
#
# Arguments:
# w -       The text window in which to insert the string
# s -       The string to insert (usually just a single character)
# ----------------------------------------------------------------------

proc multitext_TextInsert { w s } {
    global gNewTag
    if {$s == ""} { return }
    set W [file rootname $w]
    upvar #0 $W data

    set usernum $data(local)
    set env $data(environment)
    set lock [multitext_CheckForLock $W $w]
    if { $lock == "locked" } { return }

    #
    # KMA: there's an insertion so see if we're between an insert mark
    # If you try to insert text between a deletion, the cursor moves to
    # the start of the deletion.

    set thistag [multitext_GetContainingTag $w insert_$usernum INS]

    $w tag configure $thistag -underline 1

    multitext_SetTagColour $w $thistag $data(colour)

    if { $gNewTag } {
       $env set data.$thistag.pos MARK
       $env set data.$thistag.text DEFINED-IN-WIDGET
       $w mark set $thistag insert_$usernum
       $w mark gravity $thistag left
    }

    $w insert insert_$usernum $s $thistag
    $w see insert

    #
    # extend our locking extension if we're inside our lock

    if { $lock == "mylock" } {
       $w tag add lock_$usernum [$w index "insert_$usernum -              \
          [string length $s]c"] insert_$usernum
    }

    multitext_TypingTo                                                  \
       multitext_DoTextInsert $w $usernum $s $thistag $gNewTag
    multitext_SetCursor $w [$w index insert_$usernum] right
}

# ---------------------------------------------------------------------
# Actually do the text insertion
# ---------------------------------------------------------------------

proc multitext_DoTextInsert {w usernum s tagname newtag} {
    set W [file rootname $w]
    upvar #0 $W data

    if { $newtag } {
       $w mark set $tagname insert_$usernum
       $w mark gravity $tagname left
    }

    $w insert insert_$usernum $s $tagname
    set colour [gk_viewsGetAttribute $data(-scrollid) $usernum color]
    multitext_SetTagColour $w $tagname $colour
    $w tag configure "$tagname" -underline 1
}

# ---------------------------------------------------------------------
# Routines for manipulating the multiple markers
# ---------------------------------------------------------------------

proc multitext_ShiftInsertionForCursors { w pos moveto } {

    if { $moveto == "left" } { set op "-" } else { set op "+" }

    #
    # look to our left to see if there's a marker
```

```
   set leftSide [$w index "$pos - 1c"]
   set tags [$w tag names $leftSide]

   foreach i $tags {
      if [regexp "insert_*" $i] {

         #
         # there's an insertion point, so move left or right and call
         # this routine again (in case there's stacked cursors)

         set pos [$w index "$pos $op 1 char"]
         if [$w compare $pos >= [$w index "insert lineend"]] {
            set pos [$w index "insert lineend - 1 char"]
            set moveto left
         } elseif [$w compare $pos <= [$w index "insert linestart"]] {
            return [$w index "insert linestart"]
         }

         multitext_ShiftInsertionForCursors $w $pos $moveto
         return $pos
      }
   }

   return $pos
}

# -----------------------------------------------------------------------
# Set up the position of our cursor, or any remote cursors
# -----------------------------------------------------------------------

proc multitext_SetCursor { w pos moveto } {
   $w mark set insert $pos
   set pos [multitext_ShiftInsertionForCursors $w $pos $moveto]

   gkTextDoSetCursor $w [users local.usernum] $pos
   set mypos $pos
   set pos [multitext_AdaptPosForCursors $w $pos                          \
                                            [users local.usernum] "-"]
   multitext_TypingTo gkTextDoSetCursor $w [users local.usernum] $pos

   return $mypos
}

# -----------------------------------------------------------------------
# Actually create the insertion point
# -----------------------------------------------------------------------

proc gkTextDoSetCursor {w usernum pos} {
   gkTextSetMark $w $usernum $pos
   if {$usernum==[users local.usernum]} { $w see insert }

   $w mark set insert_$usernum $pos
   $w tag remove sel_$usernum 1.0 end
}

# -----------------------------------------------------------------------
# Organise a character mark for remote carets
# -----------------------------------------------------------------------

proc gkTextSetMark {w usernum posn} {

   if {$usernum != [users local.usernum]} {
      set posn [multitext_AdaptPosForCursors $w $posn $usernum "+"]
   }
```

```
    if {$usernum==[users local.usernum]} {
       $w mark set insert_$usernum $posn
       $w mark set insert $posn
       $w mark set anchor insert
    } else {
       gkTextUpdateMark $w $usernum $posn
    }
}

# ----------------------------------------------------------------------
# Build up a caret character for a remote caret
# ----------------------------------------------------------------------

proc gkTextUpdateMark {w usernum pos} {
   if {$usernum != [users local.usernum]} {
      set tags [$w tag ranges insert_$usernum]
      for {set i 0} {$i < [llength $tags]} {incr i 2} {
         $w delete [lindex $tags $i] [lindex $tags [expr $i + 1]]
      }

      $w tag delete insert_$usernum
      $w mark set insert_$usernum $pos
      $w insert $pos "\xaa" insert_$usernum
      $w tag configure insert_$usernum                               \
         -foreground [gk_getUserAttrib $usernum cursor_color]        \
         -font "*-symbol-*-*-*-10-*" -offset -10 -background pink
    }
}

# ----------------------------------------------------------------------
# Routine to actually draw the items in the widget
# ----------------------------------------------------------------------

# ---- nothing exceptional to draw

# ----------------------------------------------------------------------
# Elements of gkPriv that are used in this file:
#
# afterId -      If non-null, it means that auto-scanning is underway
#               and it gives the "after" id for the next auto-scan
#               command to be executed.
# char -         Character position on the line;  kept in order
#               to allow moving up or down past short lines while
#               still remembering the desired position.
# mouseMoved -   Non-zero means the mouse has moved a significant
#               amount since the button went down (so, for example,
#               start dragging out a selection).
# prevPos -      Used when moving up or down lines via the keyboard.
#               Keeps track of the previous insert position, so
#               we can distinguish a series of ups and downs, all
#               in a row, from a new up or down.
# selectMode -   The style of selection currently underway:
#               char, word, or line.
# x, y -         Last known mouse coordinates for scanning
#               and auto-scanning.
#
# ----------------------------------------------------------------------


# gkTextBind --
# This procedure below invoked the first time the mouse enters a text
# widget or a text widget receives the input focus.  It creates all of
# the class bindings for texts.
#
# Arguments:
# event -   Indicates which event caused the procedure to be invoked
#           (Enter or FocusIn).  It is used so that we can carry out
#           the functions of that event in addition to setting up
#           bindings.

proc gkTextBind {w event} {
   global gkPriv tk_strictMotif

   bind gkTextTag <Enter> {break}
   bind gkTextTag <FocusIn> {break}

   # Standard Motif bindings:

   bind gkTextTag <1> {
      gkTextButton1 %W %x %y ; %W tag remove sel 0.0 end }
   bind gkTextTag <B1-Motion> {
      set gkPriv(x) %x ; set gkPriv(y) %y ; gkTextSelectTo %W %x %y }
   bind gkTextTag <Double-1> {
      set gkPriv(selectMode) word ; gkTextSelectTo %W %x %y
      catch {%W mark set insert sel.first} }
   bind gkTextTag <Triple-1> {
      set gkPriv(selectMode) line ; gkTextSelectTo %W %x %y
      catch {%W mark set insert sel.first} }
   bind gkTextTag <Shift-1> {
      gkTextResetAnchor %W @%x,%y ; set gkPriv(selectMode) char
      gkTextSelectTo %W %x %y }
   bind gkTextTag <Double-Shift-1> {
      set gkPriv(selectMode) word ; gkTextSelectTo %W %x %y }
   bind gkTextTag <Triple-Shift-1>   {
      set gkPriv(selectMode) line ; gkTextSelectTo %W %x %y }
   bind gkTextTag <B1-Leave> {
      set gkPriv(x) %x ; set gkPriv(y) %y ; gkTextAutoScan %W }
   bind gkTextTag <B1-Enter> { tkCancelRepeat }
   bind gkTextTag <ButtonRelease-1> { tkCancelRepeat }
   bind gkTextTag <Control-1> { %W mark set insert @%x,%y }
   bind gkTextTag <Left> {
      multitext_SetCursor %W [%W index "insert - 1c"] left }
   bind gkTextTag <Right> {
      multitext_SetCursor %W [%W index "insert + 1c"] right }
   bind gkTextTag <Up> {
      multitext_SetCursor %W [gkTextUpDownLine %W -1] left }
   bind gkTextTag <Down> {
      multitext_SetCursor %W [gkTextUpDownLine %W 1] }
   bind gkTextTag <Shift-Left> {
      gkTextKeySelect %W [%W index {insert - 1c}] }
   bind gkTextTag <Shift-Right> {
      gkTextKeySelect %W [%W index {insert + 1c}] }
   bind gkTextTag <Shift-Up> {
      gkTextKeySelect %W [gkTextUpDownLine %W -1] }
   bind gkTextTag <Shift-Down> {
      gkTextKeySelect %W [gkTextUpDownLine %W 1] }
   bind gkTextTag <Control-Left> {
      multitext_SetCursor %W [%W index {insert - 1c wordstart}] left }
   bind gkTextTag <Control-Right> {
      multitext_SetCursor %W [%W index {insert wordend}] right }
   bind gkTextTag <Control-Up> {
      multitext_SetCursor %W [gkTextPrevPara %W insert] left }
   bind gkTextTag <Control-Down> {
      multitext_SetCursor %W [gkTextNextPara %W insert] left }
   bind gkTextTag <Shift-Control-Left> {
      gkTextKeySelect %W [%W index {insert - 1c wordstart}] }
   bind gkTextTag <Shift-Control-Right> {
      gkTextKeySelect %W [%W index {insert wordend}] }
   bind gkTextTag <Shift-Control-Up> {
```

```
   gkTextKeySelect %W [gkTextPrevPara %W insert] }
bind gkTextTag <Shift-Control-Down> {
   gkTextKeySelect %W [gkTextNextPara %W insert] }
bind gkTextTag <Prior> {
   multitext_SetCursor %W [gkTextScrollPages %W -1] left }
bind gkTextTag <Shift-Prior> {
   gkTextKeySelect %W [gkTextScrollPages %W -1] }
bind gkTextTag <Next> {
   multitext_SetCursor %W [gkTextScrollPages %W 1] left }
bind gkTextTag <Shift-Next> {
   gkTextKeySelect %W [gkTextScrollPages %W 1] }
bind gkTextTag <Control-Prior> { %W xview scroll -1 page }
bind gkTextTag <Control-Next> { %W xview scroll 1 page }
bind gkTextTag <Home> {
   multitext_SetCursor %W [%W index {insert linestart}] left }
bind gkTextTag <Shift-Home> {
   gkTextKeySelect %W [%W index {insert linestart}] }
bind gkTextTag <End> {
   multitext_SetCursor %W {insert lineend} left }
bind gkTextTag <Shift-End> { gkTextKeySelect %W {insert lineend} }
bind gkTextTag <Control-Home> {
   multitext_SetCursor %W 1.0 }
bind gkTextTag <Control-Shift-Home> { gkTextKeySelect %W 1.0 }
bind gkTextTag <Control-End> {
   multitext_SetCursor %W {end - 1 char} right }
bind gkTextTag <Control-Shift-End> {
   gkTextKeySelect %W {end - 1 char} }
bind gkTextTag <Tab> {
   multitext_TextInsert %W \t ; focus %W ; break }

bind gkTextTag <Shift-Tab> {
   # Needed only to keep <Tab> binding from triggering;  doesn't
   # have to actually do anything.
}

bind gkTextTag <Control-Tab> { focus [tk_focusNext %W] }
bind gkTextTag <Control-Shift-Tab> { focus [tk_focusPrev %W] }
bind gkTextTag <Control-i> { multitext_TextInsert %W \t }
bind gkTextTag <Return> { multitext_TextInsert %W \n }

bind gkTextTag <Delete> { multitext_TextDelete %W }
bind gkTextTag <BackSpace> { multitext_TextDelete %W }

bind gkTextTag <Control-space> { %W mark set anchor insert }
bind gkTextTag <Select> { %W mark set anchor insert }
bind gkTextTag <Control-Shift-space> {
   set gkPriv(selectMode) char ; gkTextKeyExtend %W insert }
bind gkTextTag <Shift-Select> {
   set gkPriv(selectMode) char ; gkTextKeyExtend %W insert }
bind gkTextTag <Control-slash> { %W tag add sel 1.0 end }
bind gkTextTag <Control-backslash> { %W tag remove sel 1.0 end }

gkTextClipboardKeysyms $w F16 F20 F18
bind gkTextTag <Insert> {
   catch {multitext_TextInsert %W [selection get -displayof %W]} }
bind gkTextTag <KeyPress> { multitext_TextInsert %W %A }

# Ignore all Alt, Meta, and Control keypresses unless explicitly
# bound. Otherwise, if a widget binding for one of these is
# defined, the <KeyPress> class binding will also fire and insert
# the character, which is wrong.  Ditto for <Escape>.

bind gkTextTag <Alt-KeyPress> {# nothing }
bind gkTextTag <Meta-KeyPress> {# nothing}
bind gkTextTag <Control-KeyPress> {# nothing}
bind gkTextTag <Escape> {# nothing}

# Additional emacs-like bindings:

if !$tk_strictMotif {
   bind gkTextTag <Control-a> {
      multitext_SetCursor %W [%W index {insert linestart}] left }
   bind gkTextTag <Control-b> {
      multitext_SetCursor %W [%W index insert-1c] left }
   bind gkTextTag <Control-d> "multitext_TypingTo gkTextDoDelete \
      $w [users local.usernum] insert_[users local.usernum]"
   bind gkTextTag <Control-e> {
      multitext_SetCursor %W [%W index {insert lineend}] right }
   bind gkTextTag <Control-f> {
      multitext_SetCursor %W [%W index insert+1c] right }
   bind gkTextTag <Control-k> { gkTextDeleteLine %W }
   bind gkTextTag <Control-n> {
      multitext_SetCursor %W [gkTextUpDownLine %W 1] left }
   bind gkTextTag <Control-o> {
      %W insert insert \n ;%W mark set insert insert-1c }
   bind gkTextTag <Control-p> {
      multitext_SetCursor %W [gkTextUpDownLine %W -1] left }
   bind gkTextTag <Control-t> { gkTextTranspose %W }
   bind gkTextTag <Meta-b> {
      multitext_SetCursor %W {insert - 1c wordstart} left }
   bind gkTextTag <Meta-d> { %W delete insert {insert wordend} }
   bind gkTextTag <Meta-f> {
      multitext_SetCursor %W {insert wordend} right }
   bind gkTextTag <Meta-less> {
      multitext_SetCursor %W 1.0 left }
   bind gkTextTag <Meta-greater> {
      multitext_SetCursor %W end-1c right }
   bind gkTextTag <Meta-BackSpace> {
      %W delete {insert -1c wordstart} insert }

   gkTextClipboardKeysyms $w Meta-w Control-w Control-y

   # A few additional bindings of my own.

   bind gkTextTag <Control-h> {
      if [%W compare insert != 1.0] {
         gkTextDoDelete %W [users local.usernum]                  \
                             insert_[users local.usernum]-1c
      }
   }

   bind gkTextTag <Control-v> {
       catch {
         %W insert insert [selection get -displayof %W]
         %W see insert
      }
   }

   bind gkTextTag <2> {
      %W scan mark %x %y
       set gkPriv(x) %x
       set gkPriv(y) %y
       set gkPriv(mouseMoved) 0
   }

   bind gkTextTag <B2-Motion> {
      set names [array names gkPriv]
      if {[lsearch $names x] < 0  || [lsearch $names y] < 0} {return}
      if {(%x != $gkPriv(x)) || (%y != $gkPriv(y))} {
```

```
                set gkPriv(mouseMoved) 1 }
            if $gkPriv(mouseMoved) { %W scan dragto %x %y }
        }

        bind gkTextTag <ButtonRelease-2> {
            if { [lsearch [array names gkPriv] mouseMoved] < 0 } {return}
            if !$gkPriv(mouseMoved) {
              catch {
                  %W insert insert [selection get -displayof %W]
                  gkTextSeeInsert %W
              }
            }
        }
    }

    # rename gkTextBind {}
    set gkPriv(prevPos) {}
}

# ---------------------------------------------------------------------
# gkTextClipboardKeysyms --
# This procedure is invoked to identify the keys that correspond to
# the "copy", "cut", and "paste" functions for the clipboard.
#
# Arguments:
# copy -   Name of the key (keysym name plus modifiers, if any,
#          such as "Meta-y") used for the copy operation.
# cut -    Name of the key used for the cut operation.
# paste -Name of the key used for the paste operation.
# ---------------------------------------------------------------------

proc gkTextClipboardKeysyms {w copy cut paste} {
    bind gkTextTag <$copy> {
        if {[selection own -displayof %W] == "%W" &&                    \
                            [%W tag nextrange sel 0.0 end] != ""} {
            clipboard clear -displayof %W
            clipboard append -displayof %W [selection get -displayof %W]
        }
    }

    bind gkTextTag <$cut> {
        if {[selection own -displayof %W] == "%W" &&                    \
                            [%W tag nextrange sel 0.0 end] != ""}  {
             clipboard clear -displayof %W
             clipboard append -displayof %W [selection get -displayof %W]
             %W delete sel.first sel.last
        }
    }

    bind gkTextTag <$paste> {
        catch { %W insert insert                                        \
              [selection get -displayof %W  -selection CLIPBOARD] }
    }
}

# ---------------------------------------------------------------------
# gkTextButton1 --
# This procedure is invoked to handle button-1 presses in text
# widgets.  It moves the insertion cursor, sets the selection anchor,
# and claims the input focus.
#
# Arguments:
# w -        The text window in which the button was pressed.
# x -        The x-coordinate of the button press.
# y -       The x-coordinate of the button press.
# ----------------------------------------------------------------------

proc gkTextButton1 {w x y} {
   global gkPriv

   set gkPriv(selectMode) char
   set gkPriv(mouseMoved) 0
   set gkPriv(pressX) $x

   set pos [multitext_SetCursor $w [$w index @$x,$y] left]

   $w tag add sel_[users local.usernum] $pos $pos

   if {[$w cget -state] == "normal"} { focus $w }
}

# ----------------------------------------------------------------------
# gkTextSelectTo --
# This procedure is invoked to extend the selection, typically when
# dragging it with the mouse.  Depending on the selection mode
# (character, word, line) it selects in different-sized units.  This
# procedure ignores mouse motions initially until the mouse has moved
# from one char to another or until there have been multiple clicks.
#
# Arguments:
# w -       The text window in which the button was pressed.
# x -       Mouse x position.
# y -    Mouse y position.
# ----------------------------------------------------------------------

proc gkTextSelectTo {w x y} {
   global gkPriv

   set cur [$w index @$x,$y]
   if [catch {$w index anchor}] { $w mark set anchor $cur }
   set anchor [$w index anchor]
   if {[$w compare $cur != $anchor] || (abs($gkPriv(pressX) - $x)>=3)} {
      set gkPriv(mouseMoved) 1
   }

   switch $gkPriv(selectMode) {
      char {
         if [$w compare $cur < anchor] {
            set first $cur
            set last anchor
         } else {
            set first anchor
            set last [$w index "$cur + 1c"]
         }
      }

      word {
         if [$w compare $cur < anchor] {
            set first [$w index "$cur wordstart"]
            set last [$w index "anchor - 1c wordend"]
         } else {
            set first [$w index "anchor wordstart"]
            set last [$w index "$cur wordend"]
         }
        }

      line {
         if [$w compare $cur < anchor] {
            set first [$w index "$cur linestart"]
            set last [$w index "anchor - 1c lineend + 1c"]
         } else {
            set first [$w index "anchor linestart"]
            set last [$w index "$cur lineend + 1c"]
         }
      }
   }

   if {$gkPriv(mouseMoved) || ($gkPriv(selectMode) != "char")} {
      set W [file rootname $w]
      upvar #0 $W data

      gkTextDoSelect $w [users local.usernum]                          \
         [$w index $first] [$w index $last] $data(select_colour)
      multitext_TypingTo gkTextDoSelect $w [users local.usernum]       \
         [$w index $first] [$w index $last] $data(select_colour)
   }
}

# ----------------------------------------------------------------------
# Make the selection using the appropriate background colour for the
# user
# ----------------------------------------------------------------------

proc gkTextDoSelect {w usernum first last colour} {

   $w tag remove sel_$usernum 0.0 $first
   $w tag add sel_$usernum $first $last
   $w tag remove sel_$usernum $last end
   if {$usernum==[users local.usernum]} {
      $w tag remove sel 0.0 $first
      $w tag add sel $first $last
      $w tag remove sel $last end
      update idletasks
   } else {
      $w tag configure sel_$usernum -background $colour
   }
}

# ----------------------------------------------------------------------
# gkTextKeyExtend --
# This procedure handles extending the selection from the keyboard,
# where the point to extend to is really the boundary between two
# characters rather than a particular character.
#
# Arguments:
# w -      The text window.
# index -The point to which the selection is to be extended.
# ----------------------------------------------------------------------

proc gkTextKeyExtend {w index} {
   global gkPriv

   set cur [$w index $index]
   if [catch {$w index anchor}] { $w mark set anchor $cur }
   set anchor [$w index anchor]
   if [$w compare $cur < anchor] {
      set first $cur
      set last anchor
   } else {
      set first anchor
      set last $cur
   }
```

```
    $w tag remove sel 0.0 $first
    $w tag add sel $first $last
    $w tag remove sel $last end
}

# ----------------------------------------------------------------------
# gkTextAutoScan --
# This procedure is invoked when the mouse leaves a text window
# with button 1 down.  It scrolls the window up, down, left, or right,
# depending on where the mouse is (this information was saved in
# gkPriv(x) and gkPriv(y)), and reschedules itself as an "after"
# command so that the window continues to scroll until the mouse
# moves back into the window or the mouse button is released.
#
# Arguments:
# w -   The text window.
# ----------------------------------------------------------------------

proc gkTextAutoScan {w} {
   global gkPriv

   if {$gkPriv(y) >= [winfo height $w]} { $w yview scroll 2 units
   } elseif {$gkPriv(y) < 0} { $w yview scroll -2 units
   } elseif {$gkPriv(x) >= [winfo width $w]} { $w xview scroll 2 units
   } elseif {$gkPriv(x) < 0} { $w xview scroll -2 units
   } else { return }

   gkTextSelectTo $w $gkPriv(x) $gkPriv(y)
   set gkPriv(afterId) [after 50 gkTextAutoScan $w]
}

# ----------------------------------------------------------------------
# gkTextKeySelect
# This procedure is invoked when stroking out selections using the
# keyboard.  It moves the cursor to a new position, then extends
# the selection to that position.
#
# Arguments:
# w -      The text window.
# new -   A new position for the insertion cursor (the cursor hasn't
#         actually been moved to this position yet).
# ----------------------------------------------------------------------

proc gkTextKeySelect {w new} {
   global gkPriv

   if {[$w tag nextrange sel 1.0 end] == ""} {
      if [$w compare $new < insert] { $w tag add sel $new insert
      } else { $w tag add sel insert $new }
      $w mark set anchor insert
   } else {
      if [$w compare $new < anchor] {
             set first $new
             set last anchor
      } else {
             set first anchor
             set last $new
      }
      $w tag remove sel 1.0 $first
      $w tag add sel $first $last
      $w tag remove sel $last end
   }

   $w mark set insert $new
```

```
    $w see insert
    update idletasks
}

# ---------------------------------------------------------------------
# gkTextResetAnchor --
# Set the selection anchor to whichever end is farthest from the
# index argument.  One special trick: if the selection has two or
# fewer characters, just leave the anchor where it is.  In this
# case it doesn't matter which point gets chosen for the anchor,
# and for the things like Shift-Left and Shift-Right this produces
# better behavior when the cursor moves back and forth across the
# anchor.
#
# Arguments:
# w -       The text widget.
# index -Position at which mouse button was pressed, which determines
#          which end of selection should be used as anchor point.
# ---------------------------------------------------------------------

proc gkTextResetAnchor {w index} {
    global gkPriv

    if {[$w tag ranges sel] == ""} { $w mark set anchor $index ; return }
    set a [$w index $index]
    set b [$w index sel.first]
    set c [$w index sel.last]
    if [$w compare $a < $b] { $w mark set anchor sel.last ; return }
    if [$w compare $a > $c] { $w mark set anchor sel.first ; return }

    scan $a "%d.%d" lineA chA
    scan $b "%d.%d" lineB chB
    scan $c "%d.%d" lineC chC

    if {$lineB < $lineC+2} {
        set total [string length [$w get $b $c]]
        if {$total <= 2} { return }
        if {[string length [$w get $b $a]] < ($total/2)} {
            $w mark set anchor sel.last
        } else {
            $w mark set anchor sel.first
        }
        return
    }

    if {($lineA-$lineB) < ($lineC-$lineA)} {
        $w mark set anchor sel.last
    } else {
        $w mark set anchor sel.first
    }
}

proc gkTextDeleteLine {w} {
    set usernum [users local.usernum]
    if [$w compare insert == {insert lineend}] {
        multitext_TypingTo gkTextDoDelete $w $usernum insert_$usernum
      } else {
        multitext_TypingTo gkTextDoDelete $w $usernum insert_$usernum     \
                                 [list insert_$usernum lineend]

      }
}

# ---------------------------------------------------------------------
# gkTextUpDownLine --
# Returns the index of the character one line above or below the
# insertion cursor.  There are two tricky things here.  First,
# we want to maintain the original column across repeated operations,
# even though some lines that will get passed through don't have
# enough characters to cover the original column.  Second, don't
# try to scroll past the beginning or end of the text.
#
# Arguments:
# w -    The text window in which the cursor is to move.
# n -    The number of lines to move: -1 for up one line,
#        +1 for down one line.
# --------------------------------------------------------------------

proc gkTextUpDownLine {w n} {
   global gkPriv

   set i [$w index insert]
   scan $i "%d.%d" line char
   if {[string compare $gkPriv(prevPos) $i] != 0} {
      set gkPriv(char) $char }
   set new [$w index [expr $line + $n].$gkPriv(char)]
   if {[$w compare $new == end] ||                                  \
            [$w compare $new == "insert linestart"]} {
      set new $i
   }

   set gkPriv(prevPos) $new
   return $new
}

# --------------------------------------------------------------------
# gkTextPrevPara --
# Returns the index of the beginning of the paragraph just before a
# given position in the text (the beginning of a paragraph is the first
# non-blank character after a blank line).
#
# Arguments:
# w -      The text window in which the cursor is to move.
# pos -    Position at which to start search.
# --------------------------------------------------------------------

proc gkTextPrevPara {w pos} {
   set pos [$w index "$pos linestart"]
   while 1 {
      if { (([$w get "$pos - 1 line"] == "\n") &&                   \
                  ([$w get $pos] !="\n")) || ($pos =="1.0")} {
         if [regexp -indices {^[    ]+(.)}                          \
                  [$w get $pos "$pos lineend"]dummy index] {
            set pos [$w index "$pos + [lindex $index 0] chars"]
         }
         if { [$w compare $pos != insert] || ($pos =="1.0") } {
            return $pos
         }
      }
      set pos [$w index "$pos - 1 line"]
   }
}

# --------------------------------------------------------------------
# gkTextNextPara --
# Returns the index of the beginning of the paragraph just after a given
# position in the text (the beginning of a paragraph is the first
# non-blank character after a blank line).
#
```

```
# Arguments:
# w -       The text window in which the cursor is to move.
# start -Position at which to start search.
# --------------------------------------------------------------------

proc gkTextNextPara {w start} {
   set pos [$w index "$start linestart + 1 line"]
   while {[$w get $pos] != "\n"} {
      if [$w compare $pos == end] { return [$w index "end - 1c"] }
      set pos [$w index "$pos + 1 line"]
   }

   while {[$w get $pos] == "\n"} {
      set pos [$w index "$pos + 1 line"]
      if [$w compare $pos == end] { return [$w index "end - 1c"] }
   }

   if [regexp -indices {^[     ]+(.)}                                   \
                              [$w get $pos "$pos lineend"]dummy index] {
      return [$w index "$pos + [lindex $index 0] chars"]
   }

   return $pos
}

# --------------------------------------------------------------------
# gkTextScrollPages --
# This is a utility procedure used in bindings for moving up and down
# pages and possibly extending the selection along the way.  It scrolls
# the view in the widget by the number of pages, and it returns the
# index of the character that is at the same position in the new view
# as the insertion cursor used to be in the old view.
#
# Arguments:
# w -          The text window in which the cursor is to move.
# count -    Number of pages forward to scroll;  may be negative
#             to scroll backwards.
# --------------------------------------------------------------------

proc gkTextScrollPages {w count} {
   set bbox [$w bbox insert]
   $w yview scroll $count pages
   if {$bbox == ""} {
      return [$w index @[expr [winfo height $w]/2],0]
   }
   set x [expr [lindex $bbox 0] + [lindex $bbox 2]/2]
   set y [expr [lindex $bbox 1] + [lindex $bbox 3]/2]
   return [$w index @$x,$y]
}

# --------------------------------------------------------------------
# gkTextTranspose --
# This procedure implements the "transpose" function for text widgets.
# It tranposes the characters on either side of the insertion cursor,
# unless the cursor is at the end of the line.  In this case it
# transposes the two characters to the left of the cursor.  In either
# case, the cursor ends up to the right of the transposed characters.
#
# Arguments:
# w -      Text window in which to transpose.
# --------------------------------------------------------------------

proc gkTextTranspose w {
   set pos insert
```

```
    if [$w compare $pos != "$pos lineend"] {
       set pos [$w index "$pos + 1 char"]
    }

    set new [$w get "$pos - 1 char"][$w get  "$pos - 2 char"]
    if [$w compare "$pos - 1 char" == 1.0] { return }
    $w delete "$pos - 2 char" $pos
    $w insert insert $new
    $w see insert
}
```

E.9 Shared whiteboard widgets

The widgets detailed in this section are based on those available within the GroupKit shared drawing tool. In addition to the ones provided by GroupKit, there are two new drawing objects – freehand and text – both of which are listed below.

E.9.1 Freehand drawing object

```
#
# Routines based on the Model-view-controller format to create a
# freehand line widget for the shared whiteboard
# Based on the GroupKit drawing package tool

# ----------------------------------------------------------------------
# Start / sweep / end sweep
# ----------------------------------------------------------------------

proc freehandline_start {view type x y} {
   set objid [[$view model] addobj $type]
   [$view model] setattr $objid points [list $x $y]
   [$view model] setattr $objid startpt [list $x $y]
   [$view model] setattr $objid endpt [list $x $y]

   return $objid
}
proc freehandline_sweep {view objid x y} {

   set pts [[$view model] getattr $objid points]
   set l [llength $pts]
   set endpt [lrange $pts [expr $l - 2] [expr $l - 1]]
   if { $endpt == [list $x $y] } { puts "same"; return }

   [$view model] setattr $objid points \
      [concat [[$view model] getattr $objid points] $x $y]
}
proc freehandline_end {view objid x y} {
   set pts [[$view model] getattr $objid points]
   set l [llength $pts]
   set endpt [lrange $pts [expr $l - 2] [expr $l - 1]]
   [$view model] setattr $objid endpt $endpt
}

# ----------------------------------------------------------------------
# Build up the view & position handles
# ----------------------------------------------------------------------
```

```
proc freehandline_newview {view objid} {
   freehandline_getcoords $view $objid x0 y0 x1 y1
   set id [eval "[$view canvas] create line \
      [[$view model] getattr $objid points] -smooth 1 -tags $objid"]
}

proc freehandline_createHandles {view objid} {
   freehandline_getcoords $view $objid x0 y0 x1 y1
   eval "$view addhandle $objid startpt \
      [lrange [[$view model] getattr $objid points] 0 1]"
   $view addhandle $objid endpt $x1 $y1
}

proc freehandline_repositionHandles {view objid} {
   freehandline_getcoords $view $objid x0 y0 x1 y1
   $view movehandle $objid startpt $x0 $y0
   $view movehandle $objid endpt $x1 $y1
}

proc freehandline_handleDragged {view objid handle x y} {
   cw_Debug "Can't resize freehand objects" cyan
   switch $handle {

#   startpt {[$view model] setattr $objid startpt [list $x $y]}
#   endpt   {[$view model] setattr $objid endpt [list $x $y]}

   }
}

# ----------------------------------------------------------------------
# Move or redraw the lines
# ----------------------------------------------------------------------

proc freehandline_attributeChanged {view objid attr val} {
   if { $attr == "points" } {
      [$view canvas] delete $objid
        eval "[$view canvas] create line $val -smooth 1 \
         -fill [[$view model] getattr $objid colour] -tags $objid"
        [$view canvas] bind $objid <1> "$view select replace $objid"

        update idletasks
     }
}

proc freehandline_getcoords {view objid _x0 _y0 _x1 _y1} {
   upvar $_x0 x0 $_y0 y0 $_x1 x1 $_y1 y1
   set startpt [[$view model] getattr $objid startpt]
   set endpt [[$view model] getattr $objid endpt]
   if {$startpt==""} {
      set x0 0; set y0 0
   } else {
      set x0 [lindex $startpt 0]; set y0 [lindex $startpt 1]
   }

   if {$endpt==""} {
      set x1 0; set y1 0
   } else {
      set x1 [lindex $endpt 0]; set y1 [lindex $endpt 1]
   }
}
```

E.9.2 Text drawing object

```
#
# Text widget object for the shared whiteboard
# Based on the model-view-controller from the groupkit drawing tool

# ----------------------------------------------------------------------
# Initial values for the text entry widget && authoring tool widget
# ----------------------------------------------------------------------

set gText "Text to insert"
set gOrigText "Text from the authoring tool."

# ----------------------------------------------------------------------
# Start / sweep / end sweep
# ----------------------------------------------------------------------

proc stdtext_start {view type x y} {
   global gText gOrigText

   set objid [[$view model] addobj $type]
   [$view model] setattr $objid x $x
   [$view model] setattr $objid y $y

   if { $type == "origtext" } {
      set text $gOrigText
      [$view model] setattr $objid colour black
   } else {
      set text $gText
   }

   [$view model] setattr $objid text $text
   return $objid
}

proc stdtext_sweep {view objid x y} {
   [$view model] setattr $objid x $x
   [$view model] setattr $objid y $y
}

# ----------------------------------------------------------------------
# Create a new view of the text widget item
# ----------------------------------------------------------------------

proc stdtext_newview {view objid} {
   stdtext_getcoords $view $objid x0 y0 x1 y1
   if { [[$view model] getattr $objid text] == "" } {

      #
      # if the haven't got the text details, wait for them...

      after 1 [list stdtext_newview $view $objid]
   } else {
      [$view canvas] create text $x0 $y0 -justify left \
         -text [[$view model] getattr $objid text] \
         -tags [list $objid object]

      if { [[$view model] getattr $objid type] == "origtext" } {
         [$view canvas] itemconfigure $objid \
            -font "-*-new century schoolbook-bold-r-*-*-18-*"
      }
   }
}
```

```
# ---------------------------------------------------------------------
# Text handle manipulation (one in the centre of the text)
# ---------------------------------------------------------------------

proc stdtext_createHandles {view objid} {
   stdtext_getcoords $view $objid x0 y0 x1 y1
   $view addhandle $objid tl $x0 $y0
}

proc stdtext_repositionHandles {view objid} {
   stdtext_getcoords $view $objid x0 y0 x1 y1
   $view movehandle $objid tl $x0 $y0
}

proc stdtext_handleDragged {view objid handle x y} {
   set model [$view model]
   switch $handle {
      tl {$model setattr $objid x $x; $model setattr $objid y $y}
   }
}

# ---------------------------------------------------------------------
# Move the text on an x,y change
# ---------------------------------------------------------------------

proc stdtext_attributeChanged {view objid attr val} {
   if {[member $attr [list x y]]} {
      stdtext_getcoords $view $objid x0 y0 x1 y1
      [$view canvas] coords $objid $x0 $y0
      $view objectmoved $objid
   }
}

proc stdtext_getcoords {view objid _x0 _y0 _x1 _y1} {
   upvar $_x0 x0 $_y0 y0 $_x1 x1 $_y1 y1
   set model [$view model]
   set x0 [$model getattr $objid x]; if {$x0==""} {set x0 0}
   set y0 [$model getattr $objid y]; if {$y0==""} {set y0 0}
   set x1 $x0
   set y1 $y0
}
```

Appendix F

Collaborwriter utility functions

The utility code consists of libraries of tools and procedures which are used between the Collaborwriter application elements.

F.1 General purpose utility functions

```
# cw_utils.tcl
# Various utility routines used in Collaborwriter
# Latest version: 24th Jan 1996

#
# a global counter for making any component unique (in this local app)
# (e.g. a dialog box) NOTE: counter NOT unique across all conf. apps!

set gCounter 0
set gRndSeed [pid]

# ----------------------------------------------------------------------
# Return a unique number for a global counter, incrementing it as this
# routine gets called
# ----------------------------------------------------------------------

proc cw_GetCounter {} {
   global gCounter; return [incr gCounter]
}

# ----------------------------------------------------------------------
# Return a random number
# ----------------------------------------------------------------------

proc cw_Random {} {
   global gRndSeed
   set gRndSeed [expr 30903 * \($gRndSeed & 65535\) + \($gRndSeed>>16\)]
   return [string range [expr \($gRndSeed & 65535\)/65535.0] 2 end]
}

# ----------------------------------------------------------------------
# Return a unique number (based on the time + a random number...)
# ----------------------------------------------------------------------

proc cw_GetUniqueNum {} {
   set num [exec date +%j%S]
   return "$num[cw_Random]"
}

# ----------------------------------------------------------------------
# Return the maximum of two numbers
# ----------------------------------------------------------------------

proc cw_Max { a b } { return [expr ($a > $b) ? $a : $b] }
```

```
# --------------------------------------------------------------------
# Return the minimum of two numbers
# --------------------------------------------------------------------

proc cw_Min { a b } { return [expr ($a < $b) ? $a : $b] }

# --------------------------------------------------------------------
# Routine to fire a command to a specific list of users
# --------------------------------------------------------------------

proc gk_toUsers { listOfUsers args } {
   cw_Debug "msg to: $listOfUsers  -- $args" white
   foreach user $listOfUsers {
      eval gk_toUserNum $user $args
   }
}

# --------------------------------------------------------------------
# Fire an event request to the session manager (from a conference app)
# --------------------------------------------------------------------

proc cw_toRC { event } {
   dp_RDO [registrar fd] gk_postEvent $event
}

# --------------------------------------------------------------------
# Routine to make a union of 2 people lists so there are no duplicates.
# Each element consists of (<name status>). If status == leader then it
# is kept
# --------------------------------------------------------------------

proc cw_UnionPeopleLists { list1 list2 } {
   set newlist {}
   foreach i $list1 {
      set name [lindex $i 0]
      set status [lindex $i 1]
      set idx [cw_ListSearch $list2 0 $name]
      if { $idx != -1 } {
         if {[lindex [lindex $list2 $idx] 1] == "leader"} {
            set status "leader"
         }
         set list2 [lreplace $list2 $idx $idx]
      }
      lappend newlist [list $name $status]
   }
   return [concat $newlist $list2]
}

# --------------------------------------------------------------------
# Routine to reverse a list (useful to order a keyed lists where things
# are added to the front)
# --------------------------------------------------------------------

proc cw_ReverseList { origlist } {
   set newlist {}
   foreach i $origlist {
      set newlist [linsert $newlist 0 $i]
   }
   return $newlist
}

# --------------------------------------------------------------------
# Routine to act like a console window
# --------------------------------------------------------------------
```

```
proc cw_Debug { msg colour } {
   if { ![winfo exists .debugwindow] } {
      toplevel .debugwindow
      text .debugwindow.l -yscrollcommand ".debugwindow.sb set"            \
         -background black -width 50 -height 5                              \
         -font *-lucida-medium-r-*-*-10-*-*-*-*-*-*
      scrollbar .debugwindow.sb -orient vertical                            \
         -command ".debugwindow.l yview"
      pack .debugwindow.sb -side right -fill y
      pack .debugwindow.l -side left -fill both -expand yes
   }

   set tag [cw_GetCounter]
   .debugwindow.l insert end "$msg\n" $tag
   .debugwindow.l tag configure $tag -foreground $colour
   .debugwindow.l yview end
}

# --------------------------------------------------------------------------
# Environment transaction mechanism - no changes to the environment are
# notified to others (or ourself) until the transaction is committed
# --------------------------------------------------------------------------

gk_notifier environmentTransaction
environmentTransaction bind transaction "cw_ReceiveTransaction %X %N %P"

# --------------------------------------------------------------------------
# Initialise the new environment commands and override the set command
# --------------------------------------------------------------------------

proc cw_InitialiseTransaction { envName } {
   $envName command set begintransaction "cw_BeginTransaction"
   $envName command set sendtransaction "cw_SendTransaction"
   $envName command rename set _cwtransset
   $envName command set set "cw_SetTransaction"
}

# --------------------------------------------------------------------------
# If in a transaction, just add the change to a packet of data for
# later transmission
# --------------------------------------------------------------------------

proc cw_SetTransaction { env cmd node data } {
   if { [$env option get transaction.name] == "" } {
      $env _cwtransset $node $data
   } else {
      set packet [$env option get transaction.packet]
      lappend packet [list $node $data]
      $env option set transaction.packet $packet
   }
}

# --------------------------------------------------------------------------
# Initialise the transaction data structure
# --------------------------------------------------------------------------

proc cw_BeginTransaction { env cmd transaction } {
   if { [$env option get transaction.name] != "" } {
      puts "Transaction is already set for environment $env"
      return ""
   } else {
      $env option set transaction.name $transaction
      return $transaction
   }
}
```

```
# ---------------------------------------------------------------------
# Fire the transaction - use the gk_toAll & notifier object to send it
# to all linked processes
# ---------------------------------------------------------------------

proc cw_SendTransaction { env cmd } {
   if { [$env option get transaction.name] == "" } {
      puts "No transaction to fire for environment $env"
      return ""
   } else {

      #
      # how do I copy the transaction to all - use an event?? Need to
      # ensure another notification isn't triggered though!

      gk_toAll environmentTransaction notify transaction              \
         [list [list X $env]                                          \
         [list N [$env option get transaction.name]]                  \
         [list P [$env option get transaction.packet]]]
      $env option delete transaction
   }
}

# ---------------------------------------------------------------------
# Receive notification of the transaction. Set a flag so that further
# notifications arn't sent and finally set the transactionName item to
# trigger any other linked events
# ---------------------------------------------------------------------

proc cw_ReceiveTransaction { env transactionName packet } {
   set origState [$env option get inhibit_notify]
   set origExpect [$env option get expect_update]
   $env option set inhibit_notify yes
   $env option set expect_update yes
   $env option set transaction.name ""
   foreach p $packet {
      $env set [lindex $p 0] [lindex $p 1]
   }
   $env option set expect_update $origExpect
   $env option set inhibit_notify $origState
   $env set $transactionName [$env get $transactionName]
   puts "Transaction $transactionName sent to $env"
}

# ---------------------------------------------------------------------
# Environment locking mechanism - the whole environment gets locked.
# Race conditions may still occur, unless a get/set is performed on one
# specific machine
# ---------------------------------------------------------------------

# ---------------------------------------------------------------------
# Initialise environment commands: needs only be called once when the
# environment is created
# ---------------------------------------------------------------------

proc cw_InitialiseLock { envName } {
   $envName command set lock "cw_LockEnvironment"
   $envName command set unlock "cw_UnlockEnvironment"
   $envName command rename set _cwset
   $envName command set set "cw_SetEnvironment"
}

# ---------------------------------------------------------------------
# On a attempt to perform a 'set' operation, first check if the
# environment is locked before doing the command. If locked, return '0'
# otherwise return the locking user's name
# -------------------------------------------------------------------------

proc cw_SetEnvironment { env cmd node data } {
   set locks [$env keys locks]
   foreach lock $locks {
      if { [string first $lock $node] == 0 } {

         #
         # we've found a lock, so see who has locked it

         set lockedBy [$env get $lock]
         if { $lockedBy != "[userprefs name]" } {
            puts "$env $node locked by $lockedBy"
            return 0
         }
         break
      }
   }

   $env _cwset $node $data
}

# -------------------------------------------------------------------------
# Attempt to lock the environment, returning the name of who has/gets
# the lock.
# If a.b.c is locked, trying to lock a.b.c.d is disallowed, as is trying
# to lock a or a.b
# Locking a.c is allowed, as is locking a.b.d
# -------------------------------------------------------------------------

proc cw_LockEnvironment { env cmd lockItem userID } {

   #
   # if we are the unique user ID, deal with the lock request, otherwise
   # pass it on to the unique user

   if { [_gk_getUniqueUser] == [users local.usernum] } {
      set locks [$env keys locks]
      foreach lock $locks {
         if { (![string first $lock $lockItem]) ||                     \
            (![string first $lockItem $lock])} {
            set lockedBy [$env get $lock]
            puts "$env is already locked by $lockedBy"
            return $lockedBy
         }
      }
      $env set locks.$lockItem $userID
      return $userID
   } else {
      return                                                            \
         [gk_toUserNum [_gk_getUniqueUser] $env $cmd $lockItem $userID]
   }
}

# -------------------------------------------------------------------------
# Unlock the environment, checking that we had the original lock
# -------------------------------------------------------------------------

proc cw_UnlockEnvironment { env cmd lockItem userID } {

   #
   # if we are the unique user ID, deal with the lock request, otherwise
   # pass it on to the unique user
```

```
    if { [_gk_getUniqueUser] == [users local.usernum] } {
       set lockItem [lsearch [$env keys locks] $lockItem]
       if { $lockItem != -1 } {
          set lockedBy [$env get locks.$lockItem]
          if { $lockedBy == $userID } {
             $env delete locks.$lockItem
             return $userID
          } else {
             puts "$env is not locked by you, but by $lockedBy"
             return $lockedBy
          }
       }
       return ""
    } else {
       return                                                             \
          [gk_toUserNum [_gk_getUniqueUser] $env $cmd $lockItem $userID]
    }
}

# ---------------------------------------------------------------------
# Given a list of lists, return the toplevel list element number
# matching 'item' at position 'elementNum' in the sublist
# ---------------------------------------------------------------------

proc cw_ListSearch { list elementNum item } {
   set n 0
   foreach i $list {
      if { [lindex $i $elementNum] == $item } { return $n }
      incr n
   }

   cw_Debug "cw_ListSearch returning -1" magenta
   return -1
}

# ---------------------------------------------------------------------
# Routines for browsing files that are linked to a text entry box
# ---------------------------------------------------------------------

proc cw_BrowseFiles { entryToChange showBitmaps } {
   FSBox "Select file..." "" [list cw_BrowseFilesOK $entryToChange]     \
      "" $showBitmaps
}

proc cw_BrowseFilesOK { entryToChange } {
   global fsBox
   set name "$fsBox(internalPath)/$fsBox(name)"
   cw_SetEntry $entryToChange $name
}

# ---------------------------------------------------------------------
# Given a filename, parse it to see if there are any environment
# variables inside it. If there are, get them expanded and returned
# ---------------------------------------------------------------------

proc cw_ExpandFileName { f } {
   global env
   set i [string first "$" $f]

   if { $i >= 0 } {
      set stringLen [string length $f]
      set newf [string range $f 0 [expr $i -1]]

      while { $i < $stringLen } {
         set chr [string index $f $i]
```

```
        if { $chr == "$" } {
            set end [string wordend $f [expr $i + 1]]
            set envVar [string range $f [expr $i + 1] [expr $end - 1]]
            if [info exists "env($envVar)"] {
                set replacement "$env($envVar)"
                set newf "$newf$replacement"
            } else {
                set newf [string range $f $i [expr $end - 1]]
            }
            set i $end
        } else { set newf "$newf$chr"; incr i }
    }
  } else { set newf $f }

  return $newf
}

# ---------------------------------------------------------------------
# Insert a substring into another string at index "place"
# ---------------------------------------------------------------------

proc cw_StringInsert { s place newString } {
   return "[string range $s 0 $place]$newString[                         \
      string range $s [expr $place+1] end]"
}

# ---------------------------------------------------------------------
# Return the number of instances of 'ch' in string 's'
# ---------------------------------------------------------------------

proc cw_NumberOf { s ch } {
   set numCases 0

   set found [string first $ch $s]
   while { $found != -1 } {
      incr numCases
      set s [string range $s [expr $found + 1] end]
      set found [string first $ch $s]
   }

   return $numCases
}

# ---------------------------------------------------------------------
# Given a name (eg. desktop.icons.keith) convert it so that any dots
# are replaced with "_"
# ---------------------------------------------------------------------

proc cw_ChangeDotInName name {
   set name [split $name "."]
   return [join $name "_"]
}

# ---------------------------------------------------------------------
# Given a name (eg. team name) convert it so that any spaces are removed
# The space is replaced with an "_" character instead.
# If lowercase is set to "1" the string is also made lowercase
# (for widget use)
# ---------------------------------------------------------------------

proc cw_RemoveSpacesFromName { name lowercase } {
   regsub -all " " $name "_" newName
   if { $lowercase == 1 } { return [string tolower $newName] }
   return $newName
}
```

```
# ---------------------------------------------------------------------------
# Given a name (eg. team name) convert it so that any spaces are added.
# The "_" is replaced with a space character instead
# ---------------------------------------------------------------------------

proc cw_AddSpacesToName name {
   regsub -all "_" $name " " newName
   return $newName
}

# ---------------------------------------------------------------------------
# Create a simple dialog consisting of OK and Cancel buttons
# ---------------------------------------------------------------------------

proc cw_CreateDialog { w } {

   if [winfo exists $w] {

      #
      # if the window already exists, bring it to the front of the
      # screen

      raise $w
      return 0
   }

   toplevel $w
   button $w.ok -text "Create" -width 8
   button $w.cancel -text "Cancel" -command "cw_RemoveDialog $w"       \
      -width 8
   return 1
}

# ---------------------------------------------------------------------------
# Delete a dialog which has been on-screen (e.g. from a CANCEL or OK
# click)
# ---------------------------------------------------------------------------

proc cw_RemoveDialog dialog {
   destroy $dialog
}

# ---------------------------------------------------------------------------
# Fill in an entry field in a dialog with value "value"
# ---------------------------------------------------------------------------

proc cw_SetEntry { w value } {
   $w delete 0 end
   if { $value != "" } { $w insert 0 $value }
   $w selection clear
   $w selection range 0 end
}

# ---------------------------------------------------------------------------
# Set a button to be enabled or disabled depending on the current
# listbox selection
# ---------------------------------------------------------------------------

proc cw_ListboxBtn { linkedList linkedButton } {
   if { $linkedButton != "" } {
      if { [$linkedList curselection] == "" } { set state disabled
      } else { set state normal }
      catch { $linkedButton configure -state $state }

   }
}
```

```
# -----------------------------------------------------------------------
# Create an image, checking through the <image> types available, and
# with a specified name
# -----------------------------------------------------------------------

proc cw_CreateNamedImage { imageTypes fname imgName } {
   global collab_ImagesUsed

   set ok 0

   #
   # first check to see if the image is already loaded

   if { [info exists collab_ImagesUsed($imgName)] == 0 } {
      foreach i $imageTypes {
         if { [catch {image create $i $imgName -file $fname}] == 0 } {
            set ok 1
            break
         }
      }

      if { $ok == 1 } {
         set collab_ImagesUsed($imgName) 0
      } else {
         return 0
      }
   }

   incr collab_ImagesUsed($imgName)
   return 1
}

# -----------------------------------------------------------------------
# Create an image, with the image name being the same as the filename
# used to create it
# -----------------------------------------------------------------------

proc cw_CreateImage { imageTypes fname } {
   return [cw_CreateNamedImage $imageTypes $fname $fname]
}

# -----------------------------------------------------------------------
# Delete an image
# -----------------------------------------------------------------------

proc cw_DeleteImage { imgName } {
   global collab_ImagesUsed

   if { [info exists collab_ImagesUsed($imgName)] == 0 } { return }
   incr collab_ImagesUsed($imgName) -1

   if { $collab_ImagesUsed($imgName) == 0 } {
      catch { image delete $imgName }
      unset collab_ImagesUsed($imgName)
   }
}

# -----------------------------------------------------------------------
# Environment sharing toggles
# Groupkit provides the ability to have environments sharing data. It
# is an all-or-nothing approach, however, and doesn't allow for the
# selective sharing of data at specific times.
# This routine, however, solves this problem!
#
# Based largely on the environment tcl routines in groupkit
```

```
#
# To use, after creating an environment (make sure it is initally
# shared), use the following:
#   <environment> option set coupling <none | tight>
#
# ----------------------------------------------------------------------

proc cw_InitialiseCoupling {env} {
   $env command rename set cw_gkSet
   $env command rename delete cw_gkDelete

   $env command set set cw_CoupledEnvSet
   $env command set delete cw_CoupledEnvDelete

   #
   # need the following line so that our changes get broadcast to the
   # world (if they're put as pending they never get sent ????

   $env option set expect_update no
}

proc cw_CoupledEnvSet {env cmd key val} {
   set coupling [$env option get coupling]

   if { $coupling == "tight" } {
      $env cw_gkSet $key $val
   } else {
      $env _doset $key $val
   }
}

proc cw_CoupledEnvDelete {env cmd key val} {
   set coupling [$env option get coupling]

   if { $coupling == "tight" } {
      $env cw_gkDelete $key $val
   } else {
      $env _dodelete $key $val
   }
}

proc cw_CoupleEnvironment {env} {
   set coupling [$env option get coupling]
   if { ($coupling == "none") || ($coupling == "forcenone") } {
      gk_toAll catch [list $env option set coupling tight]

      #
      # ??? should I fire a msg to get all the fields updated ???

   } elseif { $coupling == "tight" } {
      cw_Debug "Coupling is already tight for environment: $env" white
   } else {
      cw_Debug "ERROR: Coupling was $coupling" white
   }
}

proc cw_UncoupleEnvironment {env} {
   set coupling [$env option get coupling]
   if { $coupling == "tight" } {
      gk_toAll catch [list $env option set coupling none]

      #
      # ??? how can i delete the updateentrant binding ???

   } elseif { $coupling == "none" } {
      cw_Debug "Coupling is already off for environment: $env" white
   } else {
      cw_Debug "ERROR: Coupling was $coupling" white
   }
}

# -----------------------------------------------------------------------
# Pop up a Not Implemented Yet dialog
# -----------------------------------------------------------------------

proc cw_NotImplementedYet { args } {
   tk_dialog .niy "Not Implemented Yet"                                  \
      "This feature has not been implemented yet.\n$args" warning 0 OK
}
```

F.2 Generic tree manipulation functions

```
# -----------------------------------------------------------------------
# cw_tree.tcl
# Tree manipulation routines for Collaborwriter
#
#
# These routines add commands to an environment structure for tree
# manipulation.
# -----------------------------------------------------------------------

# cwtree_InitialiseTree <envname> should be called first by the calling
# program.
#
# Commands added are:
#   addbelow <parent> <item> - adds an item below parent. If item
#                              already exists, its child elements
#                              are also moved to their new location.
#   getparent <item>         - find the parent element for <item>,
#                              return "" if none.
#   getparents <item>        - return a list of nodes which are
#                              parents of <item>
#   getposition <item>       - return [depth breadth] position of
#                              item, -1 if none.
#   dotoall <item> <cmd> ??  - execute <cmd> on <item> and all its
#                              child nodes.

# -----------------------------------------------------------------------
# Initialise the tree environment
# -----------------------------------------------------------------------

proc cwtree_InitialiseTree { envName } {
   $envName command set addbelow "cwtree_AddItemBelow"
   $envName command set getposition "cwtree_GetItemPosition"
   $envName command set getparent "cwtree_GetParent"
   $envName command set dotoall "cwtree_DoToAll"
   $envName command set getparents "cwtree_GetParents"
}

# -----------------------------------------------------------------------
# Environment commands: dotoall
# usage:  envname dotoall someitem somecommand [args]
#   where one of the args is a '?' character which is replaced with
#   the appropriate <item>
# -----------------------------------------------------------------------

proc cwtree_DoToAll { env cmd item cmdToDo args } {
   set changer [lsearch $args "?"]
   _cwtree_DoToAll $env $item $cmdToDo $changer $args
}

proc _cwtree_DoToAll { env item cmdToDo changer params } {
   set thisItem [cwtree_FindItem $env $item]
   set children [$env keys $thisItem]

   set params [lreplace $params $changer $changer $item]
   eval "$cmdToDo $params"

   foreach i $children {
      _cwtree_DoToAll $env $i $cmdToDo $changer $params
   }
}

# -------------------------------------------------------------------
# Environment commands: getposition
# usage:  set depth&breadth [envname getposition someitem]
#          where depth is from 1 (root) and breadth is from 1
#         (left to right ordering)
# -------------------------------------------------------------------

proc _cwtree_GetItemPosition { env item path key coords } {
   if { $key == $item } { return $coords }

   set depth [lindex $coords 0]
   set breadth 0
   incr depth

   set keys [cw_ReverseList [$env keys $path]]

   foreach i $keys {
      incr breadth
      set state [_cwtree_GetItemPosition $env $item "$path.$i" $i      \
         [list $depth $breadth]]
      if { $state != -1 } { return $state }
   }

   return -1
}

proc cwtree_GetItemPosition { env cmd item } {
   return [_cwtree_GetItemPosition $env $item tree "" [list 0 0]]
}

# -------------------------------------------------------------------
# Environment commands: getparent
# usage:  set parent [envname getparent someitem]
#          where parent is the item found less its leaf node
# -------------------------------------------------------------------

proc cwtree_FindParent { env item } {
   set path [cwtree_FindItem $env $item]
   set parent [file rootname $path]
   return $parent
}

proc cwtree_GetParent { env cmd item } {
   return [cwtree_FindParent $env $item]
}

# -------------------------------------------------------------------
# Environment commands: addbelow
# usage:  envname addbelow someparent someitem
# where any items below 'someitem' also get moved to under 'someparent'
# -------------------------------------------------------------------
```

```
proc _cwtree_MoveChildren { env item toPath } {
   $env set $toPath leaf

   set children [cw_ReverseList [$env keys $item]]
   foreach i $children {
      _cwtree_MoveChildren $env "$item.$i" "$toPath.$i"
   }
}

proc cwtree_AddItemBelow { env cmd below item } {
   set parent [cwtree_FindParent $env $below]
   set this [cwtree_FindItem $env $item]

   if {$below == ""} { set path "$parent.$item" } else {
      set path "$parent.$below.$item" }

   if { $this != "tree" } {

      #
      # the add has resulted in moving some data in the tree, so move
      # any data under the node to its new position

      $env set $path leaf
      _cwtree_MoveChildren $env $this $path
      $env delete $this
   } else {
      $env set $path leaf
   }
}

# -------------------------------------------------------------------------
# FindItem
# Search the tree for <item>, returning its environment path structure
# e.g FindItem hello -> tree.level1.level2.hello
# -------------------------------------------------------------------------

proc _cwtree_FindItem { env item path key } {
   if { $key == $item } { return $path }

   set keys [$env keys $path]
   foreach i $keys {
      set state [_cwtree_FindItem $env $item "$path.$i" $i]
      if { $state != "tree" } { return $state }
   }
   return "tree"
}

proc cwtree_FindItem { env item } {
   return [_cwtree_FindItem $env $item tree ""]
}

# -------------------------------------------------------------------------
# Environment commands: getparents
# usage:  set parentList [envname getparents someitem]
#   where parentList is a list of parents (1st item is the root node)
# -------------------------------------------------------------------------

proc cwtree_GetParents { env cmd item } {
   set path [cwtree_FindItem $env $item]
   set parents [lrange [split $path "."] 1 end]
   if { $parents != "" } { set parents [lreplace $parents end end] }
   return $parents
}
```

F.3 Scrolling button lists utility functions

```
# -----------------------------------------------------------------------
# cw_boxscroll.tcl
# Checkbox scrolling routine for Collaborwriter
# Used by various apps to enable scrolling rows of buttons
# Developed because listboxes were lots of hassle to manipulate
# Also has additional aspect where local attributes are shown in red
# while inherited attributes are shown in black text
# -----------------------------------------------------------------------

# -----------------------------------------------------------------------
# Kick the project buttons into being updated
# -----------------------------------------------------------------------

proc boxscroll_MoveCheckButtons { dlog index vars } {
   upvar #0 $vars mvars

   if { $index == $mvars(listIndex) } { return }
   set mvars(listIndex) $index

   for { set i 0 } { $i < $mvars(listSize) } { incr i } {
      set menuCmd $mvars(menutable.[expr $i +1].cmd)
      for { set j 1 } { $j < $mvars(numAttrs) } { incr j } {
         set state $mvars(stateArray$j-[expr $index + $i])
         if {$state != $mvars(cbArray$j-$i)} {
            $menuCmd $vars $j $i $state $dlog.f.p.f$i 0
         }
      }

      if { $mvars(typeArray[expr $index + $i]) == "local" } {
         set colour red
      } else {
         set colour black}

      for { set j 1 } { $j < $mvars(numAttrs) } { incr j } {
         $dlog.f.p.f$i.cb$j-$i configure -fg $colour
      }
   }
}

# -----------------------------------------------------------------------
# Called as the result of a scroller drag - we need to 'scroll' our
# checkbuttons too
# -----------------------------------------------------------------------

proc boxscroll_ListChanged { vars dlog cmd cmd2 args } {
   upvar #0 $vars mvars

   eval "$cmd $cmd2 $args"
   set topy [lindex [$dlog.f.n.namelist yview] 0]
   set topIndex [expr int($mvars(totalMembers) * $topy)]
   boxscroll_MoveCheckButtons $dlog $topIndex $vars
}

# -----------------------------------------------------------------------
# Routine to change a button value by changing its linked array
# -----------------------------------------------------------------------

proc boxscroll_ChgBtnVal { vars col row val w fromMenu } {
   upvar #0 $vars mvars
   set mvars(cbArray$col-$row) $val
   set mvars(stateArray$col-[expr $mvars(listIndex) + $row]) $val
}
```

```
# ---------------------------------------------------------------------
# Generic 'update' routine for resetting and redrawing the checkbuttons
# after new entrants have arrived
# ---------------------------------------------------------------------

proc boxscroll_UpdateDialog { vars dlog } {
   upvar #0 $vars mvars

   set oldSize $mvars(listSize)
   set mvars(listSize) [cw_Min $mvars(totalMembers) 5]

   if { $mvars(listSize) != $oldSize } {

      #
      # the list has changed size so we need to add/delete
      # some checkbuttons

      if { $mvars(listSize) > $oldSize } {
         for {set i $oldSize} {$i < $mvars(listSize)} {incr i}{
            frame $dlog.f.p.f$i
            pack $dlog.f.p.f$i
            set state normal

            for {set j [expr $mvars(numAttrs)-1]} {
               $j > 0} {incr j -1} {
               set m $dlog.f.p.f$i.cb$j-$i
               menubutton $m -pady 0 -bd 1 -relief raised
               -width 6 -indicatoron 1 -menu $m.m                      \
               -textvariable "{$dlog}mvars(cbArray$j-$i)"

                menu $m.m -tearoff 0
                set menuItems $mvars(menutable.$j.items)
                set menuCmd $mvars(menutable.$j.cmd)

                foreach k $menuItems {
                  $m.m add command -label $k -command                  \
                      [list $menuCmd $vars $j $i $k $dlog.f.p.f$i 1]
                }

                $menuCmd $vars $j $i $mvars(stateArray$j-$i)           \
                      $dlog.f.p.f$i 0
                pack $m -side right
            }
         }
      } else {
          for {set i $oldSize} {$i > $mvars(listSize)} {incr i -1} {
            destroy $dlog.f.p.f[expr $i - 1]
         }
      }

      #
      # now we need to set the listbox height and jump to the first
      # element

      $dlog.f.n.namelist configure -height $mvars(listSize)
      $dlog.f.n.namelist yview 0
      set idx 0
   } else {
      set idx $mvars(listIndex)
   }

   #
   # we just need to force the checkbuttons into being updated

   set mvars(listIndex) -1
   boxscroll_MoveCheckButtons $dlog $idx $vars
}
```