GroupKit Tutorial

Mark Roseman
Saul Greenberg
Salaam Yitbarek

Department of Computer Science
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Contact: Saul Greenberg
Phone: +1 403 220-6087
E-mail: saul@cpsc.ucalgary.ca

May, 1994, v2.2a of GroupKit

Contents

Introduction

Overview: GroupKit Architecture

Tutorials

Tutorial One Running a GroupKit Application

Tutorial Two Hello World: A GroupKit Conference Application

Tutorial Three Background Concepts

Tutorial Four Minimalist Brainstormer

Tutorial Five Brainstormer

Tutorial Six Other Examples of GroupKit Applications

Reference

The .groupkitrc Resource FIle

Setting and Getting User Attributes and Information

Multi-casting remote procedure calls

The Keyed List Data Structure

Widgets

The GroupKit Menu Bar

Show Participants

The “About GroupKit” Information Window

Help Windows

The GroupKit Scroll Bar

Telepointers

Introduction

GroupKit Overview

GroupKit is a toolkit for developing real-time groupware applications. Based on Berkeley's public domain Tcl/Tk language, it provides the basis for building groupware applications. Tcl is an interpreted shell-type programming language, and Tk is an interface toolkit for the X11 window system. GroupKit also relies on Tcl-DP, a Tcl front-end to standard Unix sockets, for its communication needs. GroupKit programs therefore run on Unix machines.

About this Tutorial

This tutorial is an introduction to developing groupware applications using GroupKit. Having completed this tutorial, the reader should be able to run and develop groupware applications using many of the important concepts and components provided by the toolkit. However, the tutorial does not show how to construct registrar clients (which most people will not want to do anyways, at least not right away!)

Previous Knowledge

This tutorial (and the remainder of the GroupKit documentation) assumes a familiarity with Tcl/Tk, as well as some knowledge about groupware systems and their features. Knowledge of the Tcl-DP extension is not required unless you are dipping into GroupKit's source code. The tutorial also assumes that Tcl, Tk, TCL-DP, and GroupKit have been installed on your Unix system. If these systems have not been installed, see the README file in the GroupKit distribution.

Remainder of this Document

This tutorial describes the organization of GroupKit programs, followed by mini-tutorials on:

1. Starting GroupKit and running simple applications.
2. Walking you through a variety of GroupKit conference applications.
3. Short reference sections on concepts and GroupKit constructs.

For Further Information

The README file in the GroupKit distribution provides details for installing GroupKit on your Unix system. You should read this if you are installing and/or maintaining GroupKit.

The *GroupKit Code Insfrastructure Manual*, included in this distribution, gives detailed descriptions of GroupKit's source modules. You should read this only if you are interested in walking through the source code that makes up GroupKit.

GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications, (Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work—CSCW'92, Toronto, Ontario) describes an earlier version of GroupKit designed using C++ and InterViews, a C++ interface toolkit. It illustrates some of the important concepts in GroupKit.

Building Flexible Groupware Through Open Protocols, (Proceedings of the 1993 ACM Conference on Organizational Computing Systems—COOCS '93), describes the *open protocols* method of groupware design upon which GroupKit relies.

Tcl/Tk as a Basis for Groupware, (Proceedings of the 1993 Tcl Workshop, Berkeley, CA), explains the benefits of Tcl/Tk for groupware development.

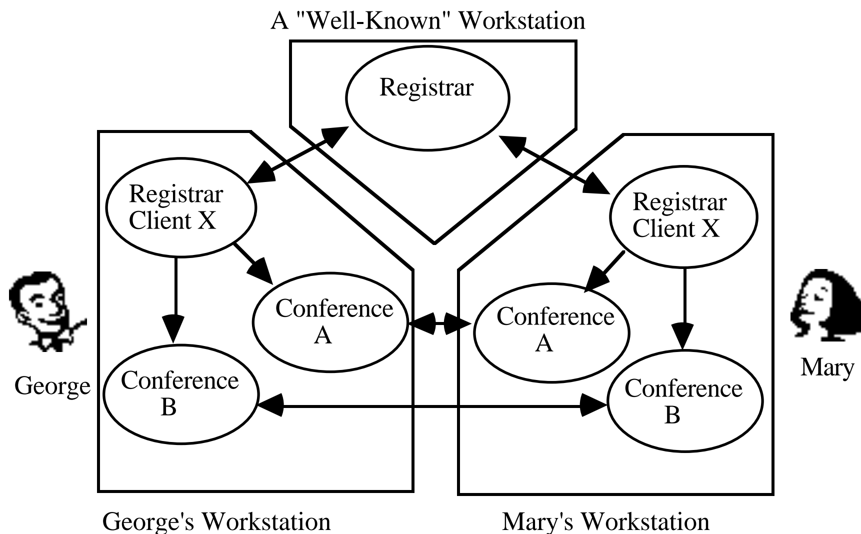
An Introduction to Tcl/Tk, (to be published by Addison-Wesley approx. Feb. '94), by John K. Ousterhout is a learning guide to Tcl/Tk.

Documentation on the Tcl-DP extension can be found in its on-line manual pages which should have been installed with Tcl-DP.

Overview: GroupKit Architecture

Overview

GroupKit applications consist of a number of mostly replicated processes arranged across a number of machines. The following diagram illustrates an example of the processes running when two people are communicating to each other through two conferences 'A' and 'B'. Each person would see windows representing each conference on their screen, as well as a window representing their Registrar Clients. Processes are represented by ovals, and the lines joining them indicate communication paths.



Registrar

The central Registrar runs on a “well-known” workstation, and is the contact point that all processes use to find out what conferences are around. It is started by executing *registrar.* The Registrar maintains a list of all the conferences active on the system and a list of all the conference users. The Registrar itself does not have a policy on how conferences are created or deleted nor on how users join or leave conferences. There is usually one Registrar for your entire community of conference users. The diagram above illustrates this.

Registrar Client

The Registrar Client (one per user) allows users to create, delete, join or leave conferences. It interacts with other Registrar Clients through the central list provided by the Registrar. The Registrar Client provides both a user interface as well as a policy dictating how conferences are created or deleted and how users are to enter and leave conferences. Different Registrar Clients can be created to suit different registration needs. Three Registrar Clients—*open.reg, command.reg,* and *end_conf.reg*—are provided in the GroupKit package. Developers can create new registrar clients if desired, although this is not covered by this tutorial. The diagram above shows how each person has a registrar client on their workstation.

Conference Applications

The conference application is the actual shared application built using GroupKit, and is separate from the registration system. A conference application is typically a groupware tool like a shared editor, whiteboard, and so on. The GroupKit application developer creates these conference applications.

The diagram above illustrates these concepts. Conference A (perhaps a shared whiteboard) is made up of two replicated processes, one belonging to Registrar Client X running on George's machine and seen by George, and the other to Registrar Client Y that runs on Mary's machine and seen by her. These conference processes maintain communication facilities necessary for exchanging messages with each other. The user interface portions of groupware applications use these facilities extensively.

Tutorial One—Running a GroupKit Application

This mini-tutorial describes how an end-user would set up and run GroupKit applications. We shall run th *Minimalist Brainstormer,* a simple brainstorming tool that allows users to enter ideas in an entry box and put them into a shared listbox. By the end of this tutorial, you should be able to run any existing GroupKit application.

The first step to running a GroupKit application is installing GroupKit. Follow the instructions in the README file that comes with the GroupKit package to install GroupKit.

The .groupkitrc file

You should now have all the GroupKit files in place, and a *.groupkitrc* file in your home directory. This *.groupkitrc* file should have been configured with you as the user.

The *.groupkitrc* file contains some information on the user and the host machine of the registrar. It also contains a list of GroupKit applications that are currently available and their paths. A sample *.groupkitrc* file is shown below (for brevity, it probably differs somewhat from the one found in the distribution).

```
# Information about yourself: Your full name
set gk_local_user(name) "Joe Bloggs"

# Information about yourself: Your color---different users can have different colors!
set gk_local_user(color) red

# Information about your network:  Set this to your internet domain
set gk(internet_domain) ".cpsc.ucalgary.ca"

# Information about your network: Set this to the computer that will run the registrar
set gk(registrar_host_name) janu

# Example GroupKit Applications
# A Hello World example in GroupKit
set gk_program(Hello\ World)    \
        "exec gkwish -f $gk_library/confs/hello_world.tcl"

# A very simple brainstorming tool
set gk_program(Minimalist\ Brainstormer) \
        "exec gkwish -f $gk_library/confs/minimalist_brainstorm.tcl"

# A very simple multi-user sketchpad
set gk_program(Simple\ Sketchpad)       \
        "exec gkwish -f $gk_library/confs/simple_sketchpad.tcl"

# A usable brainstorming tool that updates new participants, has
# a multi-user scroll bar, allows saving of idea, resizes properly, etc
set gk_program(Brainstormer)     \
        "exec gkwish -f $gk_library/confs/brainstorm.tcl"

# A File Viewer
set gk_program(File\ Viewer)    \
        "exec gkwish -f $gk_library/confs/file_viewer.tcl"

# A simple multi-user talk program
set gk_program(Text\ Chat)      \
        "exec gkwish -f $gk_library/confs/text_chat.tcl"

# A Tic Tac Toe game
set gk_program(Tic\ Tac\ Toe)   \
        "exec gkwish -f $gk_library/confs/tic-tac-toe.tcl"
```

Note that there is an established, system-wide location for GroupKit applications—**gk_library**—specified when GroupKit is installed. However, applications can also be fetched from other directories if the full path is specified. You can also over-ride this location, or lessen your dependence on it, by setting the Unix environement variable **GK_LIBRARY** in your .cshrc or similar place. For example, if the gk_library (or a copy of it) was located in /home/me/lib/groupkit, you would add this to your .cshrc: ***setenv GK_LIBRARY /home/me/lib/groupkit***

The array indices of **gk_program** (**File\ Viewer, Brainstormer** etc.) can be any string, although white space must be escaped with a '\'. They need not be the same as the names of the programs they represent. HOWEVER, they must be the same in the *.groupkitrc* files of users who wish to participate in the same conferences.

Running an Application

Having completed the preliminary tasks above, we can run a GroupKit application listed in the *.groupkitrc* file.

a) Starting a Registrar

The first step is to create a registrar process. This is done only once on the designated machine. The machine name should be the same one set in the .groupkitrc in the variable gk(registrar_host_name). You normally have only one registrar process that is used by all your GroupKit users. Do this by logging on to that machine and entering

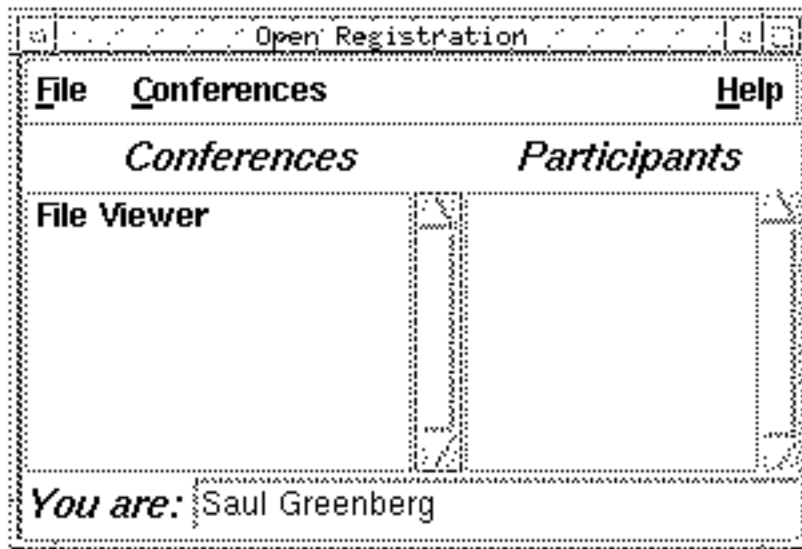
registrar &

b) Starting a registrar client

Next, start a registrar client. We shall use *open.reg,* the basic registrar client supplied in GroupKit. Enter:

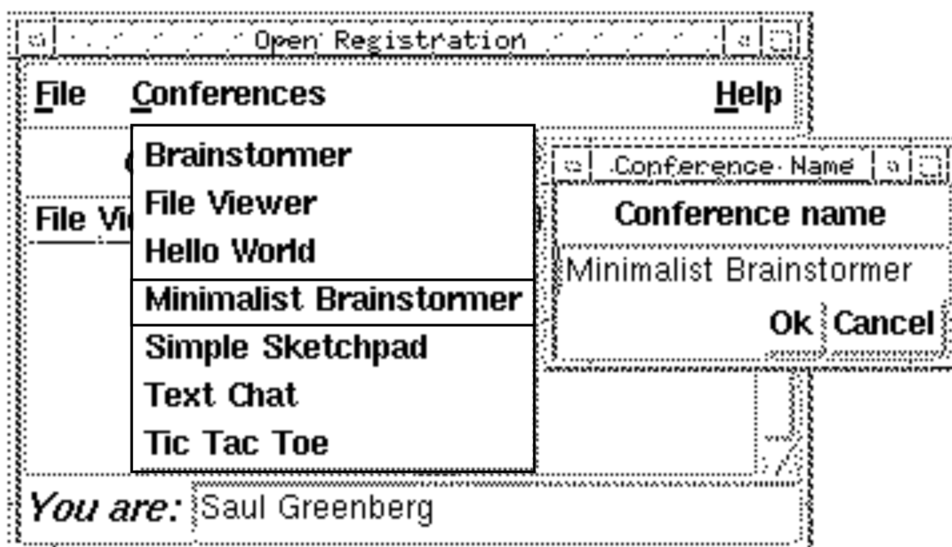
open.reg &

This brings up the following window. For illustration, we show it with one conference application called "File Viewer" running; your own conferences window will probably be empty for now. Your name is shown on the bottom, and you can change this if you wish. The name is the same one set in the .groupkitrc by the variable **gk_local_user(name).**



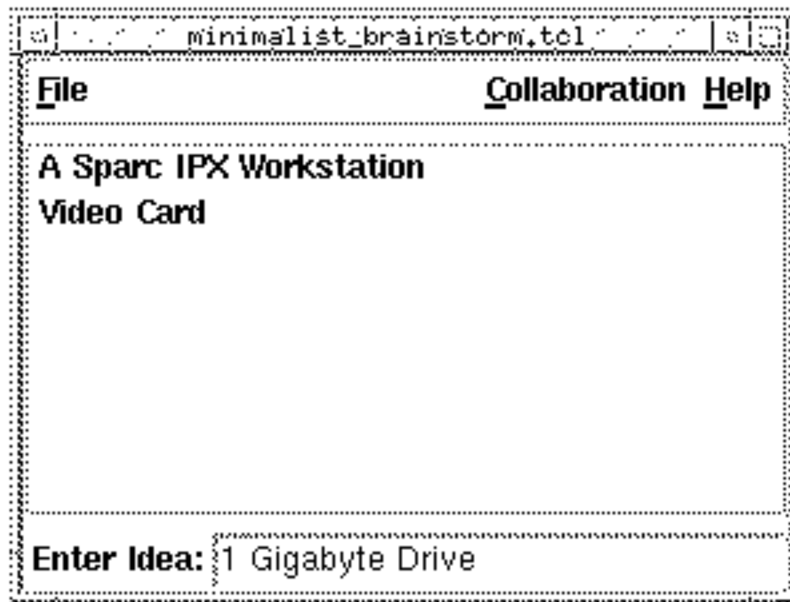
c) Starting a conference application

To start a new conference, pull down the "Conferences" menu and choose one of the listed conference applications (the list of applications are specified in the *.groupkitrc* file). When you select one, e.g., "Tic Tac Toe", a dialog box will ask you to confirm; it will also let you optionally change the name of the conference from the name of the application to something that is more meaningful to the targetted group e.g., "Equipment Purchase Ideas". Here is what the "Conferences" pull-down menu and the dialog box look like:



This starts up *Minimalist Brainstormer* conference, whose code was supplied with GroupKit in *confs/minimalist_brainstorm.tcl*, that will run in a new window displayed on your screen.

d) Running a simple brainstorming tool



You should see a window something like this (although this one shows it after a few "ideas" have been entered). To enter ideas, just type into the Enter Idea field. When you hit the RETURN key, the idea will be added to the shared list shown in the middle of the window.

You will see that the name of the conference is now displayed in the "Conferences" list box in the openrc.

e) Joining a Conference

At this point, you are just running the Minimalist Brainstormer as a single-user application. You can invite someone else to create an open.reg registrar client on their machine, or you can pretend that there are two users in the conference by creating another open.reg on your own machine.

The other user should click once on Minimalist Brainstormer in the Conferences list (not the menu!) to display the list of users of that conference in the "Users" listbox. Double-clicking on the item will automatically join that conference.

There are now two replicated processes running as a "Minimalist Brainstormer" conference. Ideas may be entered from either process and they will appear in both shared lists.

f) Exiting simplebrain

To exit from a conference, choose "Quit" under the "File" menu. This eliminates the local conference process. You can also see who is in the conference by selecting "Show Participants" in the "Collaboration" pull-down menu, and get information about GroupKit and help on this application by selecting from the "Help" pull-down menu.

Limitations

Minimalist Braintormer is a very simple GroupKit application. It does not contain facilities for saving and loading files, and more importantly, does not update new entrants into the conference. You may have noticed that the ideas you typed before the second openrc joined the conference did not appear in the new conference process! A more complicated brainstorm application, *Brainstormer*, contains the above and other features and is also included in the GroupKit release.

Summary

You should now be able to run any existing GroupKit application following the steps listed above. You should also be able to change your *.groupkitrc* file so that it only includes the conference applications that you want. In order to learn how to write a GroupKit application, see the next mini-tutorial.

Tutorial Two—Hello World: A GroupKit Conference Application

Now that you know how to run GroupKit conference applications, we will show you how to build some of your own. First will be *Hello World*, a groupware version of the “hello world” examples common to many programmin languages (you can find the source in $gk_library/confs/hello_world.tcl). It is a good program for seeing some really basic groupware code, but it is not powerful enough to be really useful. Later tutorials will formally introduce terminology, show more complex applications.

Before going on, try running the *Hello World* conference application with several participants (see Tutorial One for an explantion on how to run a conference). As you will see, *Hello World* includes the GroupKit default menu and associates some help with the *Help* pulldown menu, and displays a button labelled “Hello World” (the first screen below). When you press the button, the button label changes on all participants displays (including yours) to “Saul Greenberg says hello!” (or whatever your name is) (the second screen below). This is truly a hello world program!

And here is the entire code, with an explanation on the next page.

```
# Make the window resizable, initialize the conference, and add the GroupKit menu bar
wm minsize . 250 50
gk_initConf $argv
gk_defaultMenu .menubar
pack .menubar -side top -fill x

# Define the help and add it to the menu bar
set help_title "About Hello World"
set help_text  { {normal}  {Press the button to say hello to all conference participants.}}
gk_helpAddTopicToMenu .menubar $help_title $help_text

# Text displayed by the buttons; greeting message says who it is from
set standard_message "Hello World"
set greetings_message [concat [gk_getLocalAttrib username] "says hello!"]

# Groupware callback: Briefly post the greeting, then revert back to the standard message
proc say_hi {new_message} {global standard_message
    .hello configure -text $new_message
    after 1500 {.hello configure -text $standard_message}
}

# Create our "hello world" button and put it all together
button .hello \
    -text $standard_message \
    -command "gk_toAll say_hi [list $greetings_message]"
pack .hello -side top
```

Hello world, like all conference applications, begins with the procedure call **gk_initConf $argv**. This call establishes the run-time infrastructure around the application. We also include a standard tk instruction to the window manager that makes the window resizable. We then add in GroupKit’s default menu bar which can be obtained by way of a call to **gk_defaultMenu**, with the parameter **.menubar** which will become the menu bar’s widget name.

The next step is to add help to the Help pulldown. First we define a help title and some formatted help text (see the widget reference section for an explanation of how help formatting is defined). These are supplied to **gk_helpAddTopicToMenu**, along with the menubar widget path, to automatically add an item labelled “About Hello World” to the pulldown, and to create the appropriate help window when that item is selected.

Two messages are then defined. The standard message is just “Hello World”. The second greetings message gets the name of the local user by the call **gk_getLocalAttrib** with the argument **username**, and concatenates it with the string “says hello”.

We then define a callback, called say_hi, that will briefly change the label in the button from “Hello World” to the one defined in the greeting message.

The button is then created. The novel thing about it is its command: **gk_toAll**, which tells all participant to execute the command the procedure say_hi, whose argument is the greeting message containing the local user’s name. This means that when one person (say George) presses the button, all instances of the application will execute the callback *say hi “George says hello”*, and the button is changed throughout.

Tutorial Three—Background Concepts

This section will introduce some GroupKit detail and terminology. By the end of this tutorial, you should have a basic awareness of some GroupKit concepts. While some of the concepts and their uses may not be immediately clear, you should be able to recognize them when you see them in the code examples.

Terminology

Conference terminology can be difficult to understand, so read this carefully before proceeding.

A conference in GroupKit has two meanings. In its first sense, a conference (short for *conference application*) means a shared tool used in a meeting. *Hello World,* for example, implements a conference application. Users can *join* conference applications in progress.

In its second sense, a conference means a *conference process.* In the replicated architecture used to implement a GroupKit conference application, one conference process is created for each participant. This means that if there are (say) three participants in a *Hello World* conference application, then there are three conference processes running in that application. Thus, a user participating in a conference has a conference process running on his machine that is replicated on the machines of all the other users participating in that conference.

About Users and User Attributes

Conference applications sometimes have to know about its participants (users). For example, a programmer may want to find some information (called attributes) of the local and remote users. Attributes are contained in a data structure called a "keyed list", which is just a list of name/value pairs. The routines below are completely defined in the reference section "Setting and Getting User Attributes and Information", and you should refer to that for more detail. Here is a brief summary.

gk_getRemoteUsers is a procedure that returns a keyed list of all the remote conference participants with information on each of them.

gk_getLocalUser is a procedure that returns a keyed list containing information on the local conference participant.

gk_findUser ***usernum*** is a procedure that, given a user's unique user number, returns that user's keyed list.

These keyed lists contain a variety of standard attributes that define the user (more can be added by the programmer. Several convenience routines allow attributes to be looked up.

gk_getUserAttrib ***usernum attribute*** is a procedure that, given a participant's unique user number ***usernum*** returns the value associated with that attribute.

gk_getLocalAttrib ***attribute*** is a procedure that returns the value associated with that attribute for the local user.

gk_setUserAttrib ***usernum attribute value*** is a procedure that sets the value associated with the named attribute for a particular user

gk_setLocalAttrib ***attribute value*** is a procedure that sets the value associated with the named attribute for the local user.

A few more things you can do:

gk_amOriginator is a procedure that returns 1 if you are the original creator of the conference application, otherwise 0.

Tracking the Arrival and Departure of Users in a Conference Application

The GroupKit programmer may want to find about about users when they join or leave the conference. GroupKit will automatically set some variables to reflect users arrivals and departures, which can then be traced and acted upon by the programmer. The three important variables are listed below and detailed in the reference section on "Tracking Arrivals, Departures, and Entrants".

gk_newUser is a variable that contains a keyed list containing information on a new user that had just joined a conference.

gk_deletedUser is a variable that contains a keyed list containing information on a user that had just left a conference.

gk_entrantUser is a variable, set in only one conference process, that contains a keyed list containing information on a user that had just entered a conference. Its purpose is to tell one conference process to send the entering user the current state of the conference.

Autoloading

GroupKit programs rely on a number of GroupKit library routines. These are found through the standard TCL method of autoloading instead of sourcing. Locations of GroupKit procedures are found through the tclIndex file in

$gk_library. This way, an application need not source the files which contain the GroupKit functions they wish to call. You probably won't have to worry about this unless you split your own applications across several files.

Tutorial Four—Minimalist Brainstormer

This section will walk you through another example—the *Minimalist Brainstormer*. This is just another simple example to get you comfortable; all its concepts were already shown in the Hello World example

The *Minimalist Brainstormer*, which you already saw illustrated and then ran in Tutorial One, is a very simplistic brainstorming tool that demonstrates some basic GroupKit ideas, but is too rudimentary to be much use. Tutorial Five will present something that is closer to a "real" brainstorming tool.

Minimalist Brainstormer is defined in GK_LIBRARY/confs/*minimalist_brainstorm.tcl*. It begins with **gk_initConf $argv**. This call internally initializes GroupKit's information about local and remote users**.** It also sends a message back to the registrar client telling it that a connection has been made. At this point, you could (if you wish), add code that queries GroupKit about the local and remote users (the calls are listed in Tutorial Three).

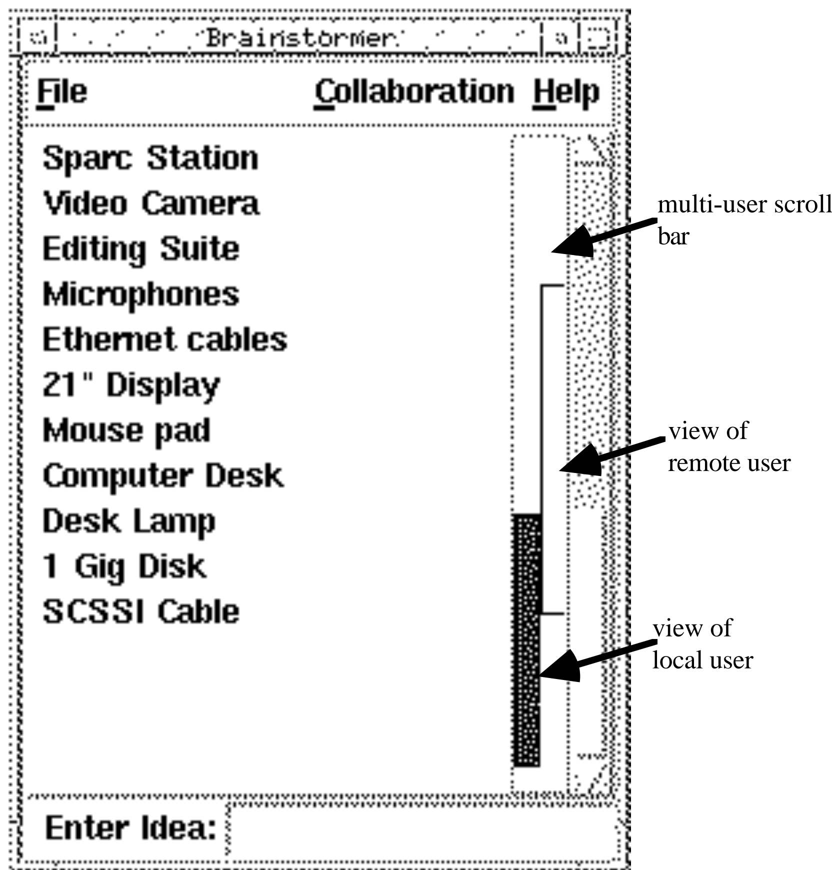
The next step is to create the interface for *minimalist_brainstorm*—the GroupKit menu bars, as well as standard TK frames, listboxes, labels, and entries. The person enters an idea (a text fragment) in the entry and hits return. This activates the callback function **put_entry_in_list**. This takes the text in the entry (the variable myIdea) and appends it to the listboxes of all conference participants. It does this by the GroupKit command **gk_toAll**, which tells all conference pocesses to execute the command is **..shared_list insert end $myIdea**.

Here is the complete program. It differs from the one in the distribution as some comments and help are omitted. Try to extend it by (for example) prefixing each idea with the name of the person who entered it.

```
# Initialize conference
gk_initConf $argv

# Tell all application instances to insert the idea into the shared list.
proc put_idea_in_list {} { global myIdea
    if {$myIdea != ""} {
        gk_toAll .shared_list insert end $myIdea
        set myIdea ""
    }
}

# Create the default groupkit pulldown menu bar
gk_defaultMenu .menubar

# Make a listbox to contain the shared ideas
listbox .shared_list -geometry 40x8 -bd 2 -relief ridge

# Make a labelled entry where the user can type in ideas.
frame .bottom
label .bottom.label -text "Enter Idea:"
entry .bottom.idea_entry -textvariable myIdea -relief sunken
focus .bottom.idea_entry

# Enter the idea into the shared list after every <return>
bind .bottom.idea_entry <Return> "put_idea_in_list"

# Put it all together
pack .bottom.label -side left
pack .bottom.idea_entry -side left -expand yes -fill x
pack .bottom -side bottom -fill x
pack .menubar -side top -fill x
pack .shared_list -side left -fill x
```

Tutorial Five—Brainstormer

This section will walk you through another example—the *Brainstormer*, whose code can be found in *brainstorm.tcl*. Here is what it looks like:



This is a fully functional version of *minimalist_brainstorm.tcl* with the following features.

1. New conference entrants are updated so that they have the same information as the existing conference participants.
2. A menu item is added to the GroupKit help menu that describes this application.
3. The listbox uses the GroupKit group scrollbar widget. The group scrollbar widget contains a local scrollbar and a remote scrollbar that shows what the remote user is looking at.
4. File handling. Ideas in the listboxes can be saved and files of ideas can be retrieved. The listboxes can also be cleared. Most is standart TCL/TK, but it does show further use of the **gk_toAll** routine.
5. The interface is more robust (in TK terms), as the windows and widgets readjust their size properly.

Although the code is longer, most of it is taken up with quite standard TK programming. The important differences are described below.

The first feature—updating new entrants—uses the GroupKit variable **gk_entrantUserNum**. This variable is updated whenever a new participant enters the conference. However, only one existing conference process—the one with the smallest user number (the oldest conference participant)—sees this. This process is the one responsible for updating the entrant by running its **updateEntrant** procedure.

In practise, applications put a trace on **gk_entrantUserNum** to call an updating function of their own. *Brainstorm.tcl* has **updateEntrant**, which sends the list of ideas in the listbox to the entrant's listbox.

The second feature—bringing up a help window—illustrates the standard way of adding help to an application; we have already seen this in *Hello World.*

The third feature is the GroupKit scrollbar widget, which can be used by any scrollable TK widget. It is actually a combination of a normal TK scrollbar and a remote scrollbar that shows the location of all the thumbs (bars) of the remote users. The remote users' thumbs are identified by a unique colour (gk_local_user(colour), if it has been set in the *.tclgkr* file). The name of the user represented by a thumb can be seen by pressing the mouse over the thumb.

The fourth feature of *brainstorm.tcl* is the file handling. It shows how menu items can be added to the GroupKit menubar, and further use of **gk_toAll.** The FSBox widget (a non-groupware Tk extension included in the GroupKit

release) is used to allow users to specify file names. Clear also shows how the group view can be maniplated, in this case by clearing the shared list of all participants when any one person selects the “Clear” option from the menu.

The code will be examined section by section. Again, it differs in minor ways from the one in the distribution.

We begin by defining the Help text. You have already seen this, except now the formatting of the help text is slightly more complex. The formated text should be a list comprised of:

{ {font type} {text} {font type} {text} ... }

and the font type is one of:

large, large_bold, large_italic, large_bold_italic
normal, normal_bold, normal_italic, or normal_bold_italic.

```
set application(title) "Brainstormer"

set application(description) {
{normal} { Type an idea into the ideas area and hit return.

The idea will display in the shared ideas list, which is scrolled\
 by the multi-user scroll bar, or by pressing the }
{normal_italic} {middle mouse button.

}
{normal} {The }
{normal_italic} {File }
{normal} {menu will let you save ideas to a file, open an existing file, or\
 clear the ideas from the display

}
{normal_bold} {Limitations:
}
{normal} {Control characters are handled poorly.}
}
```

Now we initialize the conference, and do some standard TK directives to the window manager.

```
gk_initConf $argv
wm title . $application(title)
wm minsize . 20 7
```

We then create the GroupKit pulldown menubar, and add help to it. We also show how new menu options can be added directly to other pulldown menu items (see the Reference section on the groupkit menubar).

```
# Create the default groupkit pulldown menu bar
# Then add Clear, Open, and Save items to the "File" menu, and help.
gk_defaultMenu .menubar
gk_helpAddTopicToMenu .menubar $application(title) $application(description)
.menubar$gk_menu(file_menu) add separator
.menubar$gk_menu(file_menu) add command -label "Open" -command "Open"
.menubar$gk_menu(file_menu) add command -label "Save" -command "Save"
.menubar$gk_menu(file_menu) add separator
.menubar$gk_menu(file_menu) add command -label "Clear Ideas" -command "Clear"
pack .menubar -side top -fill x
```

We create a frame, a listbox that contains the shared ideas, and a scrollbar. However, this is actually the GroupKit multiuser scrollbar, which will show the location of all participants in the scrolled document! (See the reference section on the groupkit scrollbar)

```
frame .middle -bd 2 -relief raised
listbox .middle.shared_list \
    -setgrid true \
    -yscrollcommand "gk_scroller .middle.vscroll"
gk_groupScroll .middle.vscroll -command ".middle.shared_list yview"
pack .middle -side top  -expand yes -fill both
pack .middle.vscroll -side right -fill y
pack .middle.shared_list -side left -expand yes -fill both
```

Now we create the labelled entry that the user will type into (it always has the focus), and attach a callback to it that is executed when the user hits <Return>. This is all standard TCL/TK.

```
# Create a frame at the bottom that contains a labelled entry where
# the user will type their idea.
frame .bottom
label .bottom.label \
    -text " Enter Idea:"
entry .bottom.idea_entry \
    -textvariable myIdea \
    -relief sunken
bind .bottom.idea_entry <Return> "insert_idea_into_shared_list"
focus .bottom.idea_entry

pack .bottom -side bottom -fill x
pack .bottom.label -side left
pack .bottom.idea_entry -side left -expand yes -fill x
```

All that is left is to define the procedures and callbacks. The callback insert_idea_into_shared_list is executed locally when a user enters an idea and presses <Return>. Throught the **gk_toAll** routine, the callback tells all replicated conference processes, including itself, to execute the next procedure, actually_insert_idea_into_shared_list, which actually inserts the non-empty idea into the end of all shared lists.

```
proc insert_idea_into_shared_list {} { global myIdea
    if {$myIdea != ""} {
        gk_toAll actually_insert_idea_into_shared_list $myIdea
        set myIdea ""
    }
}

proc actually_insert_idea_into_shared_list {idea} {
    .middle.shared_list insert end $idea
}
```

When a new participant enters a conference, it should be brought up to date with the current set of ideas. This is handled through **gk_entrantUserNum**, which is set in only <u>one</u> of the replicated conference applications. The callback updateEntrant tells the entering user, through the **gk_toUserNum,** to add the current ideas into its shared list one at a time.

```
trace variable gk_entrantUserNum w {updateEntrant}
proc updateEntrant args { global gk_entrantUserNum local
    for {set count 0} {$count < [.middle.shared_list size]} {incr count 1} {
        set idea [.middle.shared_list get $count]
        gk_toUserNum $gk_entrantUserNum \
            actually_insert_idea_into_shared_list $idea
    }
}
```

Finally, these procedures are executed when their corresponding pulldown menu items are selected. They save the ideas in a file, or open a new set of previously saved ideas, or clears the ideas in all praticipant's ideas list. All is standard TCL/TK, with two exceptions. First, **FSBox**, supplied with GroupKit, is a widget that presents a file selection dialog to the user and returns the result. Second, both Clear and Open use **gk_toAll** to tell all remote conference processes to clear or display a set of ideas respectively.

```
proc Save {} {
    if {[.middle.shared_list size] == 0} {return}
    set fileName [FSBox]
    if {$fileName == ""} {return}
    set fd [open $fileName a]
    for {set count 0} {$count < [.middle.shared_list size]} {incr count 1} {
        puts $fd [.middle.shared_list get $count]
    }
    close $fd
}

proc Open {} {
    set fileName [FSBox]
    if {$fileName == ""} {return}
    Clear
    set fd [open $fileName r]
    set result [gets $fd temp]
```

```
    while {$result != -1} {
        gk_toAll actually_insert_idea_into_shared_list $temp
        set result [gets $fd temp]
    }
    close $fd
}

proc Clear {} {
    gk_toAll doClear
}

# Clear a listbox.
proc doClear {} {
    .middle.shared_list delete 0 end
}
```

Tutorial 6—Other Examples of GroupKit Applications

The distribution contains several more examples that will help you learn how to develop GroupKit program.

Text Chat

Text_chat.tcl is a multi-user text-based chat program for real time conversations (similar to Unix talk). Every person has their own window, and each participant sees the text typed by all others in their respective windows. Text_chat demonstrates the following GroupKit features:

1. Participants arriving after the conference has started need to have their displays brought up to date. As with brainstorm, the variable gk_entrantUserNum is traced so that one conference process can bring the entering user up to speed.
2. Text_chat has to create a new text window for every participant. In a manner similar to **gkEntrantUserNum**, every process puts a trace on **gk_newUser**. This variable gets triggered whenever a participant enters the conference. Unlike gkEntrantUserNum, all applications are notified about the new user, and each will create a new text window on that user.
3. When a participant leaves, their text windows on the other screens have to be removed. By tracing **gk_deletedUser**, participants can trigger a callback that removes the deleted user from their display.
4. Because we want the text widget to behave differently than the normal TK widget, we completely over-ride its bindings and make any keyboard action a “groupware” action (through **insertCharacter**). In one sense, this example shows the beginnings of how a multi-user text editor would be created.

Simple sketchpad

Simple_sketchpad.tcl is an extremely simple multi-user sketchpad. It demonstrates two things.

1. How the same simple GroupKit concepts can be applied across a wide variety of domains.
2. The use of telepointers (note: these are still under development in GroupKit and will probably change in the next release). Two calls are shown:

   gk_initializeTelepointers, which initializes the use of telepointers in the application

   gk_specializeWidgetTelepointer ***widget*** which tells the application to put a telepointer in the widget specificied by **widget** (in this case the canvas) and all of its children.

File Viewer

Fileviewer.tcl is essentially the same as the brainstormer. I just added it as it is a useful application!

Tic Tac Toe

Tic_tac_toe.tcl was modelled after the tic tac toe game that was used to demonstrate the Rendezvous groupware toolkit, a (unfortunately dead) toolkit built by Ralph Hill at Bell Communications Research Labs. The game was described in various articles about Rendezvous. I wanted to see how hard it would be to replicate his interface faithfully. Turned out to be pretty easy. Anyways, look at this for another “complex” groupware application that has notions such as:

turntaking
replicated views of part of the interface that are the same for each participant
customized views that are slightly different for each participant

It could probably be cleaner, but its not bad for an afternoon’s work! See the Help menu for more information on this game.

Other applications

Other applications will be released periodically. Look in the release notes for further information, or join the groupkit mailing list by sending mail to saul@cpsc.ucalgary.ca.

Reference: The .groupkitrc Resource File

All GroupKit users must have a .groupkitrc file located in their home directory. This file contains some communication information necessary for GroupKit's run-time architecture, some information about the user, and a list of conference aliases that also specifies the how to execute the appropriate GroupKit application.

These variables should be set as described below.

Information about the user.

gk_local_user(name)
Set this to your full name

gk_local_user(color)
Set this to a legal X windows color. This color is used to represent you to other participants in some applications (e.g., Telepointers are shown in your color).

Information about your network.

gk(internet_domain)
Set this to your internet domain path.

gk(registrar_host_name)
Set this to the machine that will run the registrar.

Information about GroupKit conference applications.

gkProgram(*conference_name*)
This array should be indexed by an alias (a meaningful name) to a conference application; if there is a space in the name, it shoud be escaped with a '\'. GroupKit then takes its value and executes its. The conference name must be the same at all sites if others are to join the conference! The value is usually of the form:
exec gkwish -f path_of_groupkit_application

Example.
The following .groupkitrc file sets a user's name to Mary Smith, their color to red, to internet domain to .cpsc.ucalgary.ca, the registrar machine name to janu, and includes a link to one conference application hello_world.tcl, which is aliased to "Hello World".

```
set gk_local_user(name) "Mary Smith"
set gk_local_user(color) red

set gk(internet_domain) ".cpsc.ucalgary.ca"
set gk(registrar_host_name) janu

set gk_program(Hello\ World)    \
        "exec gkwish -f $gk_library/confs/hello_world.tcl"
```

Reference: Setting and Getting User Attributes and Information

Every user (participant) in a conference has an associated set of attributes contained in a keyed list (see Reference Section on Keyed Lists) that the programmer may need to know about. These procedures, defined in the module *conf.tcl,* allow these attributes to be queried and set.

Finding out who is in the conference

gk_getRemoteUsers
Returns the keyed lists of all the remote conference participants.

gk_getLocalUser
Returns the keyed list of all the local conference participant.

Finding out about yourself.

gk_getLocalAttrib ***AttributeName***
Get the specified attribute from the local user's set of attributes.

gk_setLocalAttrib ***AttributeName NewValue***
Set the specified attribute in the local user's set of attributes.

gk_amOriginator
Returns 1 if you are the original creator of the conference application, otherwise 0.

conftype	the type of our conference
originator	the host and port of the Registrar Client who initiated the conference
confnum	the unique id number of the conference
reghost	the host of the Registrar Client that started the conference
regport	the port of the Registar Client that started the conference
usernum	our unique user number
username	our proper name (if available) or the userid
userid	our login id

Finding out about any participant.

gk_getUserAttrib ***UserNum AttributeName***
Get the specified attribute from the specified user's set of attributes.

gk_setUserAttrib ***UserNum AttributeName NewValue***
Set the specified attribute in the specified user's set of attributes.

gk_findUser ***UserNum***
Returns the keyed list of the specified user

usernum	the unique user number for that user
userid	the login id of the user
username	the user's proper name (if available) or the userid
filedesc	the file descriptor used internally for GroupKit communications

```
proc gk_getUserAttrib {UserNum AttribName}
#  get Attribname from the list of the user with
#  usernum==UserNum
#  e.g. gk_getUserAttrib UserNum filedesc

proc gk_setUserAttrib {UserNum AttribName NewValue}
#  set AttribName in the list of the user with
#  usernum==UserNum to NewValue.
#  e.g. gk_setUserAttrib userNumber username "The Great One"

proc gk_setLocalAttrib {AttribName NewValue}
#  set AttribName in the list of the local user to NewValue.
#  e.g. gk_setLocalAttrib username "Marvelous"

proc gk_getLocalAttrib {AttribName}
#  get attribute AttribName from the local user.
#  e.g. gk_getLocalAttrib usernum

proc gk_amOriginator {}
#  Returns 1 if this conference application is the conference
#  originator, 0 otherwise.
```

Reference: Multi-Casting Remote Procedure Calls

A GroupKit programmer usually codes applications so that certain actions are executed at all replicated processes. For example, if some text is supposed to be added to a shared list, the procedure and its arguments would be invoked in all conferences. GroupKit provides several mechanisms to execute remote procedure calls (RPCs) in one, most, or all replicated processes. The ability to send many others a remote procedure call from a single call is called multi-casting:

gk_toAll ***args***
    Send a tcl command to all conference users, including self

gk_toOthers ***args***
    Send a tcl command to all conference users, EXCEPT yourself

gk_toUserNum ***user_num args***
    Send a tcl command only to the specified user

The application developer would typically use **gk_toAll** for multicasting RPCs, as it is the simplest way to invoke “universal” actions. When the developer has to send a specific RPC, for example when updating an entering user, the function **gk_toUserNum** would be used. Occasionally **gk_toOthers** is used when the local process has to do something different from the remote processes. These procedures are all in the module *conf.tcl*.

args is the remote procedure call
user_num is the number of the user to send to e.g.:
- as returned from a request to get *usernum* in user’s keyed list description
- as held by the global variables *gk_entrantUserNum*, *gk_deletedUser*, and *gk_newUser*

Examples.
The following excerpts show how these calls could be used.

```
# Insert hello into the .list widget in all conference processes
gk_toAll .list insert end "hello"

# Insert goodbye into our .list widget, and hello in to the list widget of the other
# conference processes
.list insert end "hello"
gk_toOthers .list insert end "hello"

# Tell an enterting participant to insert hello into their list widget
trace variable gk_entrantUserNum w {
       global gk_entrantUserNum
       gk_toUserNum $gk_entrantUserNum .list insert 0 hello
}
```

Reference: The Keyed List Data Structure

Keyed lists are not part of standard Tcl. They are like records in procedural programming languages—they are one structure with fields and attached values.

For example, the keyed list **book** may have the value:

{author {A. Santos}} {title {Golf}} {date {2010}}

The "keys" in this keyed list are author, title, and date. The operations that can be performed on **book** are:

keylget: Gets the value of a key. eg. *keylget book author* returns *A. Santos.*

keylset: Sets the value of a key. eg. *keylset book title Tennis* changes the title from *Golf* to *Tennis.*

keyldel: Deletes a key and its value from a list.
eg. *keyldel book author* results in the list *{title {Golf}} {date {2010}}***.**

keylkeys: *keylkeys book* returns a list of the keys in book, *{author title date}.*

Widgets. The GroupKit Menu Bar

GroupKit supplies a default menu bar that should be displayed by all GroupKit applications. The menu bar is minimally configured with two pulldown *File* and *Help* menus, and an optional *Collaboration* menu. It looks something like:

Quit allows graceful exits.
About GroupKit gives background information on GroupKit.
Show Participants shows who is in the conference.

One routine is provided to construct the menubar (implemented in the module *gk_widget_menubar.tcl)*.

gk_defaultMenu ***menuBar*** ***<display_collaborations_menu>***
Create a menubar with the path *menubar* containing the file and help pulldown menus. By default, *display_collaborations_menu* is set to “true”, and the collaboration menu is shown. If its set to false, it is not shown.

Programmers can add items to the existing menus in the standard way. The names of the file, collaboration, and help pulldown menus are defined as array elements:
gk_menu(file_menu)
gk_menu(collab_menu)
gk_menu(help_menu)

Programmers can also add new menubuttons and pulldowns to the menu bar in the standard way; the pack command will put it between the File and Collaborations menubutton.

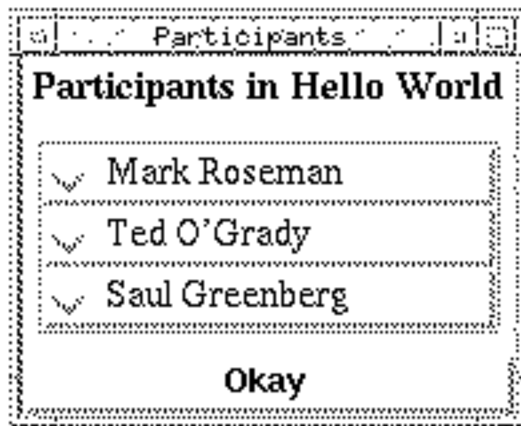
Example.
The following excerpt creates the menu bar, and adds the new item “Save” to its file menu.

```
gk_defaultMenu .menubar
.menubar$gk_menu(file_menu) add command -label "Save"
```

Widgets: Show Participants

GroupKit supplies a way to display a window of all the participants in the current conference application. The window is automatically updated as people enter and leave the conference. Depending on how the widget is called, the names of participants are either labels or radio buttons. If they are radio buttons, selecting a participant will bring up a new window that will show all the currently known attributes (as held by the keyed list for that user). This is currently useful for debugging; in a future release it will be modified to provide information that is truly useful to the end user e.g., contact information.

Here is what it looks like, showing three people participating in the conference application "Minimalist Brainstormer".



This widget is called by the *Show Participants* item in the GroupKit default menubar. Programmers can call it directly (implemented in the module gk_widget_participants.tcl).

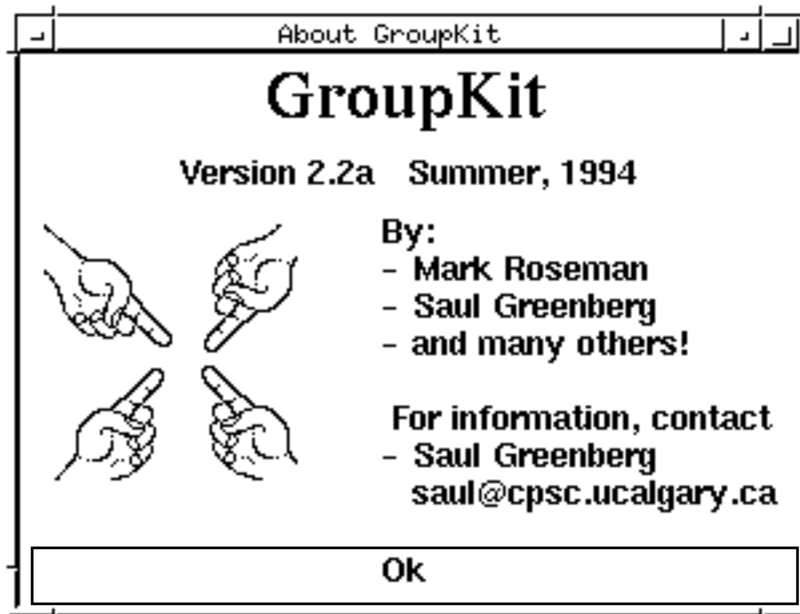
gk_showParticipants ***<show_attributes>***

Create a window that displays a continually updated list of all participants in the current conference application.

show_attributes is an optional argument; if its false (the default), participants are shown as labels. If true, they are shown as radio-buttons, which when selected will pop up a new window showing all information known about that user.

Widgets: The "About GroupKit" Information Window

GroupKit supplies an information window that describes itself i.e., version number, contact information, etc. It currently looks like:



This widget is called by the *About GroupKit* item in the GroupKit default menubar. Programmers can call it directly (implemented in the module gk_widget_about.tcl).

gk_about

Create a window that displays some information about GroupKit.

Widgets: Help Windows

Help windows.

GroupKit application should provide the user with a brief description of the application and how to use it. The user would normally select it through the "Help" pulldown menu in the groupkit menu bar.

Programmers construct help descriptions (implemented in the module *gk_widget_help.tcl)* as described here.

gk_helpAddTopicToMenu ***menubar title text***

Add a menu item labelled *title* to the groupkit menu bar. When selected, the help description, composed of a *title* and formatted *text* will be displayed in its own window.

gk_helpShowTopicInWindow ***title text***

Display a help description, composed of a *title* and some *text* in its own window on the screen. This is a programmer-controlled way of displaying the help window (i.e., instead of through the menubar).

menubar is the default groupkit menu bar that should have been created before the procedure is called.
title is a short descriptive phrase describing the help description.
text is formatted help text, a list containing:

{ {font type} {text}
{font type} {text} ...
}

The font type is one of:

large	normal
large_bold	normal_bold
large_italic	normal_italic
large_bold_italic	normal_bold_italic

Example.

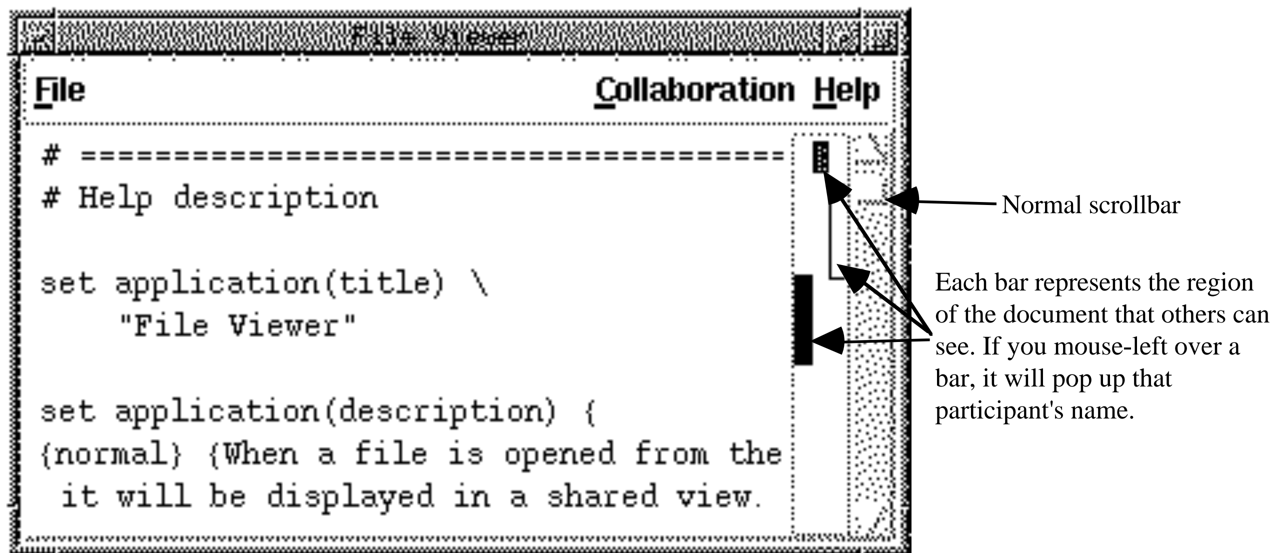
The following excerpt defines some help text, adds an item "About Simple Sketchpad" to the help menu on the menu bar. When the item is selected, a window is displayed that contains text looking like this:

```
set help_title "About Simple Sketchpad"
# Notice the formatted syntax of the text. Its ugly, but it works!
set help_text {
 {normal} {This is a very simple groupware sketchpad.

}{normal_bold} {Press } {normal} {and } {normal_bold} {drag }
 {normal_italic} {mouse button 1 }
 {normal} {to draw on the displays of \
 all participants.}
}
gk_defaultMenu .menu
gk_helpAddTopicToMenu .menu $help_title $help_text
```

Widgets: The GroupKit Scroll Bar

Warning: The GroupKit Scroll Bar is under development and will probably change in future releases. We are also aware of a small non-critical bug, where parts of the scrollbar will not update immediately when a new conference participant arrives. This usually fixes itself as soon as the new person scrolls.

GroupKit supplies a special scrollbar that not only shows your location in a view, but other people's as well. It is comprised of a standard TK scrollbar (on the left), and a special region on its right that shows a bar representing where others are and the relative size of the region they can see. If you mouse-left over a bar, it pops up the participant's name. It looks something like this:



One routine is provided to construct the scrollbar (implemented in the module *gk_widget_scrollbars.tcl)*.

gk_groupScroll ***widget_path -command "yourScrolledWidget yview"***

Create a groupkit scrollbar with the widget path *widget_path.* As with normal scrollbars, you should attach it to whatever other widget you want to scroll. Note that this does not conform to standard widgets, as this is the only syntax allowed by this call. It also only implements a vertical scrollbar. This will change in future releases.

gk_scroller ***widget_path args***

This is used to tie into the -yscrollcommand of whatever widget is being scrolled. It expects the same arguments as a normal scrollbar set command. See the example below.

Example.

The following code is a complete GroupKit application that creates a group scrollbar, a listbox with some elements, and joins them together. Type it in at try it out!

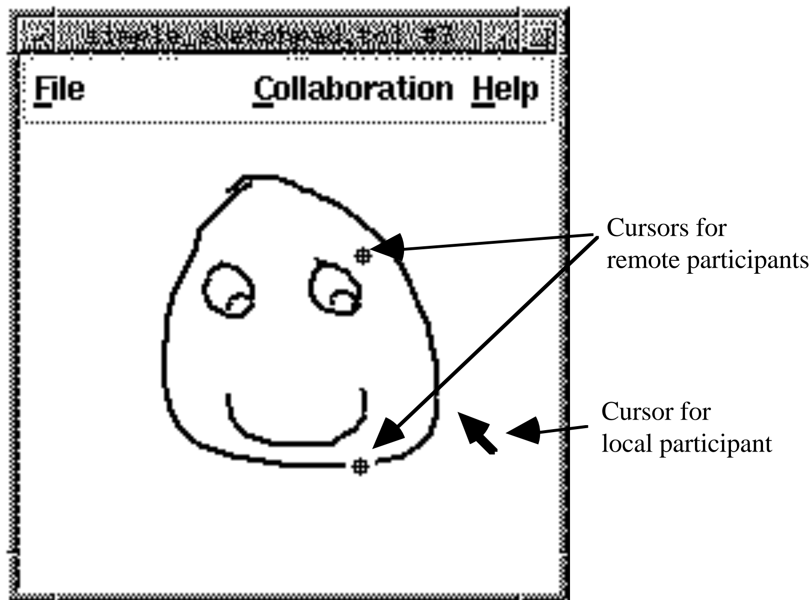
```
gk_initConf $argv

listbox .lbox  -yscrollcommand "gk_scroller .vscroll"
gk_groupScroll .vscroll -command ".lbox yview"

pack .vscroll -side right -fill y
pack .lbox -side left -expand yes -fill both

foreach item {a b c d e f g h i j k l m n o p q} {
  .lbox insert end $item
}
```

Widgets: Telepointers

Warning: The GroupKit Telepointer is under development and will probably change in future releases.

GroupKit offers the ability to add telepointers, or multiple cursors, to its applicatons on a widget by widget basis. Telepointers currently track relative to a widget, if widgets may appear in different locations on different participants' screens, it will appear in the correct place. Currently, telepointers are not smart enough to appear in the correct relative position within a scrollable widget (this will change in future releases).

For example, here is the simple sketchpad showing three people using it, showing the local cursor and two telepointers.



Several routines are provided to activate and manipulate telepointers.

gk_initializeTelepointers"
Initialize our telepointer, which also tells other participants about it. However, this doesn't actually attach it to anything.

gk_specializeWidgetTreeTelepointer ***widget***
Show the telepointer in the widget specified by ***widget*** and all of its children. The telepointer will appear relative to the widget, even if they are on different locations on other displays. In any single GroupKit application, you can thus selectively add telepointing capabilities to all, some or none of the widgets. For example, you can call this routine twice to add telepointers to (say) a canvas and a button, while ignoring the menubar.

Example.
The following complete program adds telepointers to an otherwise empty GroupKit conference.

```
gk_initConf $argv
gk_initializeTelepointers
gk_specializeWidgetTreeTelepointer .
```