

GROUPKIT TUTORIAL

Mark Roseman, Salaam Yitbarek, and Saul Greenberg
Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
Phone: +1 403 220-6087
E-mail: saul@cpsc.ucalgary.ca

December, 1993

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.

Previous Knowledge

This tutorial (and the remainder of the GroupKit documentation) assumes a familiarity with Tcl (including the Tcl-DP extension) and Tk, as well as some knowledge about groupware systems and their features. 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. Creating GroupKit conference applications.

For Further Information

The README file in the GroupKit distribution provides details for installing GroupKit on your Unix system.

The *GroupKit Reference Manual* includes detailed descriptions of the various modules which form GroupKit.

GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications, (Proceedings of CSCW '92, Toronto, Ontario) describes an earlier version of GroupKit designed using C++ and InterViews, a C++ interface toolkit. It may help illustrate some of the important concepts in GroupKit.

Building Flexible Groupware Through Open Protocols, (Proceedings of COOCS '93), describes the *open protocols* method of groupware design upon which GroupKit relies.

Tcl/Tk as a Basis for Groupware, (Proc. Tcl '93, Berkeley, CA), explains the benefits of Tcl/Tk for groupware development.

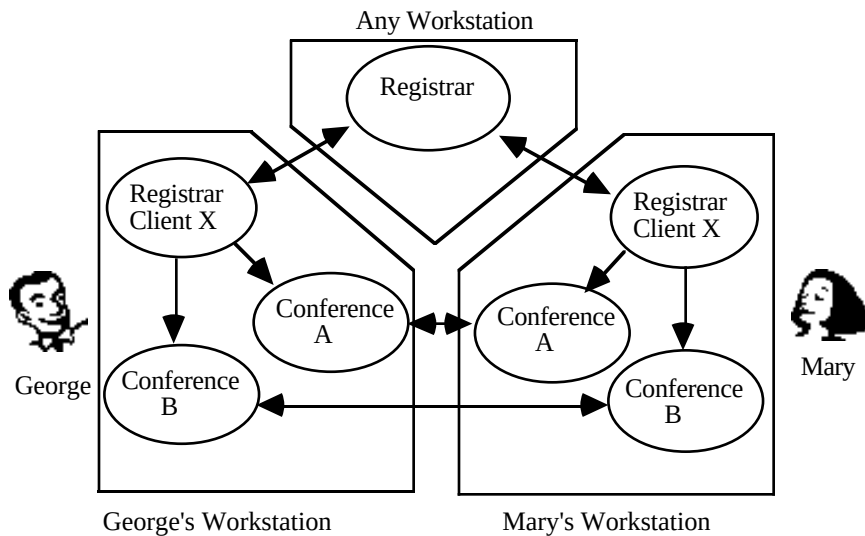
An Introduction to Tcl/Tk, (to be published by Addison-Wesley approx. Feb. '94), by John K. Ousterhout is a useful guide to Tcl/Tk.

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

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 (which may be different!). Processes are represented by ovals, and the lines joining them indicate communication paths.



Registrar

The central 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.

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—*openrc.tcl*, *cmdrc.tcl*, and *nukerrc.tcl*—are provided in the GroupKit package. Developers can create new registrar clients if desired.

Conference Applications

The conference application (conference, in short) 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.

In the diagram above, 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.

Mini-Tutorial One—Running a GroupKit Application

This mini-tutorial describes how an end-user would set up and run GroupKit applications. We shall run *simplebrain.tcl*, 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 .tclgkrc file

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

The *.tclgkrc* 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 *.tclgkrc* file is shown below (it probably differs somewhat from the one found in the distribution):

```
set myusername "Mark Roseman"
set internetdomain ".cpsc.ucalgary.ca"
set reghostname janu
# setting myColour is optional
set myColour sienna
set gk_program(GroupSketch) \
    "exec dpwish -f $GK_LIBRARY/groupsketch/groupsketch.tcl"
set gk_program(Voting\ Booth) \
    "exec dpwish -f $GK_LIBRARY/voter/vote.tcl"
set gk_program(SimpleBrain) \
    "exec dpwish -f $GK_LIBRARY/brainstorm/simplebrain.tcl"
set gk_program(Postit) \
    "exec dpwish -f $GK_LIBRARY/postit/postit.tcl"
set gk_program(Brainstorm) \
    "exec dpwish -f $GK_LIBRARY/brainstorm/brainstorm.tcl"
set gk_program(Simple\ Sketching) \
    "exec dpwish -f $GK_LIBRARY/draw/sketch.tcl"
set gk_program(Drawing\ Rectangles) \
    "exec dpwish -f /home/grads/roseman/draw/draw.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, e.g. *draw.tcl* as shown above is located elsewhere.

The array indices of **gk_program** (**GroupSketch**, **Voting\ Booth** 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 *.tclgkrc* 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 *.tclgkrc* file.

a) Starting a Registrar

The first step is to create a registrar process. This is done only once on the designated machine i.e. you normally have only one registrar process at a site.

Do this by entering

reg.sh &

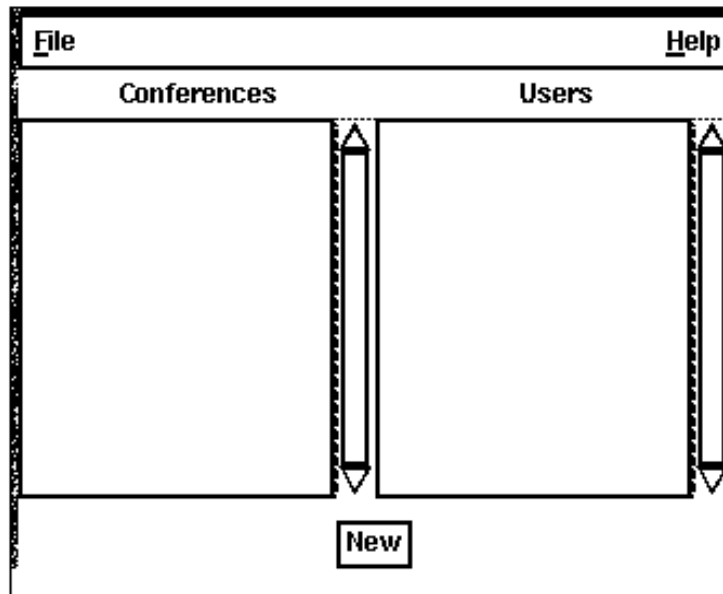
on the command line. We now have a registrar running with the computer specified by **reghostname** (in *.tclgkrc*) as its host.

b) Starting a registrar client

Next, open a registrar client. We shall use *openrc.tcl*, the basic GroupKit registrar client. Start openrc by entering

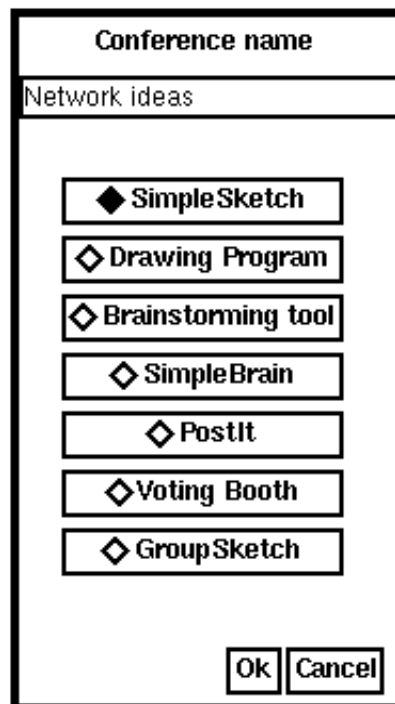
openrc.sh -hhostname &

on the command line, where **hostname** is the host machine of the registrar (this can be omitted if the host machine is reghostname). This brings up the following window:



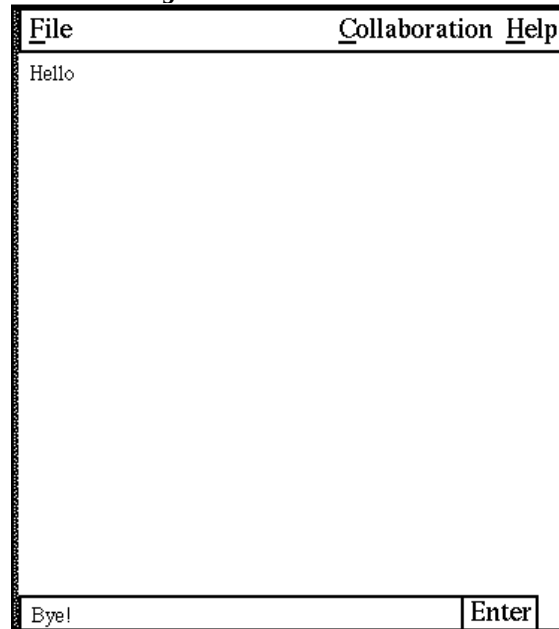
c) Starting a conference application

Select “New” to start a new conference. This brings up the conferences specified in the *.tclgkrc* file.



Enter the name by which you wish to call your conference (say, Network ideas) in the entry box (the default name is “Conference name”) and choose “SimpleBrain”. This starts up *simpleBrain.tcl*, bringing up the following window:

d) Running a simple brainstorming tool



You will see that the name of the conference, Network ideas, is now displayed in the “Conferences” list box in the openrc.

To enter ideas into the entry box at the bottom of the window, click in the box and hit the RETURN key or click on “Enter”.

e) Joining a Conference

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

The other user should click once on Network ideas to display the list of users of that conference in the “Users” listbox. Double-click on Network ideas to join the Network ideas conference.

There are now two replicated processes running as a “Network ideas conference”. Ideas may be entered from either process and they will appear in both listboxes.

f) Exiting simplebrain

To exit from a conference, choose “Quit” under the “File” menu. This eliminates the local conference process.

Limitations

simplebrain.tcl 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, *brainstorm.tcl*, contains the above and other features and is included in GroupKit.

Some of the other GroupKit conference applications, such as *postit.tcl* and *netfone.tcl*, have slightly different mechanisms for joining and communication. Read their documentation, try them, and see how you get along.

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 *.tclgkrc* 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.

Mini-Tutorial Two—Creating Conference Applications

Now that you know how to run GroupKit applications, we shall teach you how to build some of your own. We will present some very important terminology and provide some general information on conferences. We then go through two sample applications included with GroupKit:

- `minimalist_brainstorm.tcl`
- `brainstorm.tcl`

The first example is a very simplistic brainstorming tool that demonstrates some basic GroupKit ideas, but is too rudimentary to be much use. The second example is closer to a “real” brainstorming tools. By the end of this mini-tutorial, you should be able to build fairly complex GroupKit conference applications.

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. *Minimalist_brainstorm*, for example, implements a conference. Users can *join* conferences in progress.

In its second sense, a conference means a *conference process*—one process in the replicated architecture that is used to implement GroupKit applications. 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.

Autoloading

GroupKit programs rely on a number of GroupKit library routines. These are found through the standard TCL method of autoloading instead of sourcing.¹ Locations of procedures are found through the `tclIndex` files contained in library directories, which include a list of procedures and files in a directory. 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.

About Conferences Applications

All conference applications make use of the GroupKit module *conf.tcl*. The purpose of *conf.tcl* is to provide facilities for communication amongst conference processes. It does this by using its four main data structures, listed below:

`_gk_remoteUsers` is a keyed list of all the remote conference participants with information on each of them. If there are five participants in a *simplebrain.tcl* conference, each has an `_gk_remoteUsers` list with four elements.

`_gk_localUser` is a keyed list containing information on the local conference process.²

`gk_newUser` contains information on a new user joining a conference.

`gk_deletedUser` contains information on a user leaving a conference.

conf.tcl also provides procedures that allow conference processes to find information about each other.

Example 1. Creating *minimalist_brainstorm.tcl*

Minimalist_brainstorm.tcl is a “minimal” brainstorming tool. It allows the user to enter ideas into an entry box and have them displayed in a listbox and the listboxes of all the other conference participants. This is a good program for learning groupkit basics, but it is not powerful enough to be really useful.

Minimalist_brainstorm, like all conference applications, begins with the procedure call **`gk_initConf $argv`**. This call initializes **`_gk_remoteUsers`** and **`_gk_localUser`**, and sends a message back to the registrar client telling it that a connection has been made.

The next step is to create the interface for *minimalist_brainstorm*. This may include menu bars, text boxes or other widgets provided by Tk. GroupKit supplies a default menu bar which can be obtained by way of a call to **`gk_defaultMenu`**. In *minimalist_brainstorm*, we call this procedure with the parameter

¹For more on autoloading, see the Tcl library file `errors.tcl`.

²The variables begin with **`_gk_`** because they are not meant to be used directly in applications. However, applications developers should know about these variables.

.menubar. This returns a menu bar in **.menubar** that includes “File” (containing “Quit”), “Collaboration”, (containing “Show Users”) and “Help” (containing “About GroupKit”) buttons.³

The other three standard TK widgets needed in simplebrain are a listbox, an entry box, and a label (“Enter”). A 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 participant to execute the command is **..shared_list insert end \$myIdea**.⁴

Here is the complete program.

```
# A Minimalist Anonymous Brainstorming Tool
#
# Users enter "ideas", which appear on the screens of all group members.
# It is "minimalist" as a person cannot save the ideas to a file, and
# does not see old ideas if they join a session in progress.
# It is missing application-specific help, a scroll bar, and resizing
# capabilities

# Initialize conference
gk_initConf $argv

# Create the default groupkit pulldown menu bar, and put it on the top
gk_defaultMenu .menubar

# Make a listbox to contain the shared ideas, and
listbox .shared_list -geometry 40x8

# Make a labelled entry where the user can type in ideas.
frame .bottom
label .bottom.label -text Enter
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_entry_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

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

Note that this application has several limitations which are discussed in the comments at the start of the program. We overcome these limitations in *brainstorm.tcl*, which we shall create next.

³When making a customized quit or exit facility, note that the procedure used to exit gracefully from a conference is **gk_properQuit**.

⁴**gk_toAll** is a GroupKit procedure residing in the file *conf.tcl*, and is explained in the GroupKit Reference Manual.

Example 2. Creating *brainstorm.tcl*

Brainstorm.tcl 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. (This widget will be improved in later groupkit releases).
4. File handling. Ideas in the listboxes can be saved and files of ideas can be retrieved. The listboxes can also be cleared.
5. The ideas list can be cleared.
6. 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 a “standard” way to create an “About this application” message and how to add new items to existing pulldown menus.

The third feature is the GroupKit scrollbar widget, which can be used by any scrollable TK widget such as an editor or listbox. The group scrollbar is actually a frame that contains a local scrollbar (a Tk widget) 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 (myColour, if it has been set in the *.tclgkr* file). The name of the user represented by a thumb can be seen by passing the mouse over the thumb. This displays all the users represented by thumbs under the mouse. Its not really that good right now, but it will get better!

The fourth feature of *brainstorm.tcl* is the file handling. It also shows how menu items can be added to the GroupKit menubar. The FSBox Tk widget (a Tk extension included in the Groupkit release) is used to fetch the files for opening or saving.

The fifth feature shows how the group view can be manipulated, in this case by clearing the shared list of all participants when any one person selects the “Clear” option from the menu.

```
# An Anonymous Brainstorming Tool
# See the application description at the bottom of this file for details

# Initialize conference
gk_initConf $argv

# Tell the window manager the title and the default window size
wm title . "Brainstorming Tool"
wm minsize . 20 7

# Create the default groupkit pulldown menu bar
# Then add Clear, Open, and Save items to the "File" menu
gk_defaultMenu .menubar
.menubar.file.menu add separator
.menubar.file.menu add command -label "Open" -command "Open"
.menubar.file.menu add command -label "Save" -command "Save"
.menubar.file.menu add separator
.menubar.file.menu add command -label "Clear Ideas" -command "Clear"
pack .menubar -side top -fill x
```

```

# Make a frame in the window's middle with a scrollable listbox containing shared ideas.
# The twist is that the scrollbar is actually the Groupkit multiuser 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

# Make 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
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

# After a carriage return, the idea is entered into the shared list
bind .bottom.idea_entry <Return> "insert_idea_into_shared_list"

# Tell all instances of this application to insert the idea (if it
# isn't an empty string) into the shared list.
proc insert_idea_into_shared_list {} { global myIdea
    if {$myIdea != ""} {
        gk_toAll actually_insert_idea_into_shared_list $myIdea
        set myIdea ""
    }
}

# Actually insert the idea into the shared list
proc actually_insert_idea_into_shared_list {idea} {
    .middle.shared_list insert end $idea
}

# =====
# Update the entering participant with the state of the conference
# The trace on gk_entrantUserNum is only invoked for ONE of the
# existing users when a new user joins the conference.
# updateEntrant puts all the existing ideas into the new user's listbox.
trace variable gk_entrantUserNum w {updateEntrant}
proc updateEntrant args { global gk_entrantUserNum _gk_localUser
    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
    }
}

# =====
# These routines are invoked by the commands in the "File" menu.
# They save the ideas in a file, or open a new set of
# previously saved ideas, or clears the ideas in all participant's idea lists.

# Saves contents of listbox in file. The FSBox routine asks the user which
# file to save into, the file is opened and the listbox elements are put in.
# If the file already exists, the data is appendend onto it.
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
}

# Gets contents of a file and displays them, line by line, in the
# listboxes of ALL the conference processes.
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
}

# Clears ALL the listboxes.
proc Clear {} {gk_toAll doClear}

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

# =====
# These variables/routines are used to describe this application
# to the user. An "About this application" item is added to the
# Help menu, which will pop up an information window
set application(title) \
    "Brainstorming Tool"
set application(description) \
    "You can enter one-line 'ideas' by typing text followed by <return>."

Ideas of all participants,\
including yours, are displayed in the scrollable list.

Limitations:
-Control characters are handled poorly."

.menuubar.help.menu add command \
    -label "About This Application" \
    -command {about_this_applicaton}

# Create a window that describes this application to the user
proc about_this_applicaton {} {global application
    if {[wininfo_exists .about_window] == 1} { return }
    toplevel .about_window
    wm title .about_window "About this application"
    label .about_window.title \
        -text $application(title) \
        -font -Adobe-times-bold-r-normal--*-180*
    message .about_window.text -width 240 \
        -text $application(description)
    button .about_window.ok -text "Okay" -command "destroy .about_window"
    pack .about_window.title .about_window.text .about_window.ok -side top
}

```

Other examples.

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

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 override 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.

Sketch.tcl is an extremely simple multi-user sketchpad. It doesn't really show anything new, but does demonstrate how the same simple Groupkit concepts can be applied across a wide variety of domains.

Appendix: **Other Groupkit Utilities**

A few other GroupKit procedures (located in *conf.tcl*)

It is not often that an application developer would want to use `_gk_remoteUsers` or `_gk_localUser` directly. GroupKit provides procedures that send messages to remote users and that give information on particular remote users. An example of such a procedure is **`gk_toAll`**, used in the `simplebrain` and `brainstorm` example programs above. These procedures are all in the module *conf.tcl*. The relevant portion of *conf.tcl* (without the program code) is shown below:

```
proc gk_toAll args
# send a Tcl command to all conference users, including self.
# e.g.. gk_toAll .list insert end "hello"

proc gk_toOthers args
# send a tcl command to all users EXCEPT yourself.
# e.g. gk_toOthers .list insert end "hello"

proc gk_getUserAttrib {UserNum AttribName}
# get Attribname from the list of the user with
# usernum==UserNum
# e.g. gk_getUserAttrib userNumber 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_toUserNum {UserNum args}
# send a tcl command to the user with usernum==UserNum.
# e.g. gk_toUserNum userNumber myProc args

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

More on *gkwidgets.tcl*

gkwidgets.tcl provides the default menu, registrar client default menu, group scrollbars, and remote cursors widgets. These are all fully documented in the file itself, and in the GroupKit Reference Manual.

Appendix: ***Keyed Lists***

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}*.