

## GID 二次开发说明

周少林

摘要:

简要介绍 GID 软件提供的二次开发接口,并结合一个小例题来讲解接口程序的模式及方法,主要给大家一个概要的功能介绍,具体功能可以参考文章后面列出的参考文献。

关键字: GID 二次开发 TCL/TK 参数化

### 一. GID 的二次开发语言介绍

Tcl/Tk 的发明人 John Ousterhout 教授在八十年代初,是伯克利大学的教授。在其教学过程中,他发现在集成电路 CAD 设计中,很多时间是花在编程建立测试环境上。并且,环境一旦发生了变化,就要重新修改代码以适应。这种费力而又低效的方法,迫使 Ousterhout 教授力图寻找一种新的编程语言,它即要有好的代码可重用性,又要简单易学,这样就促成了 Tcl (Tool Command Language) 语言的产生。

Tcl 最初的构想的是希望把编程按照基于组件的方法 (component approach),即与其为单个的应用程序编写成百上千行的程序代码,不如寻找一个种方法将程序分割成一个个小的,具备一定“完整”功能的,可重复使用的组件。这些小的组件小到可以满足一些独立的应用程序的需求,其它部分可由这些小的组件功能基础上生成。不同的组件有不同的功能,用于不同的目的。并可为其它的应用程序所利用。当然,这种语言还要有良好的扩展性,以便用户为其增添新的功能模块。最后,需要用一种强的,灵活的“胶水”把这些组件“粘”合在一起,使各个组件之间可互相“通信”,协同工作。程序设计有如拼图游戏一样,这种设计思想与后来的 Java 不谋而合。终于在 1988 年的春天,这种强大灵活的胶水 - Tcl 语言被发明出来了。

按照 Ousterhout 教授的定义, Tcl 是一种可嵌入的命令脚本化语言 (Command Script Language)。“可嵌入”是指把很多应用有效,无缝地集成在一起。“命令”是指每一条 Tcl 语句都可以理解成命令加参数的形式:

命令 [参数 1] [参数 2] [参数 3] [参数 4] ..... [参数 N]

脚本化是指 Tcl 为特殊的,特定的任务所设计。但从现在角度看,可以说 Tcl 是一种集 C 语言灵活强大的功能与 BASIC 语言易学高效的风格于一身的通用程序设计语言。

Tk (Tool Kit) 是基于 Tcl 的图形程序开发工具箱,是 Tcl 的重要扩展部分。Tk 隐含许多 C/C++ 程序员需要了解的程序设计细节,可快速地开发基于图形界面 Windows 的程序。据称,用 Tcl/Tk 开发一个简单的 GUI 应用程序只需

几个小时，比用 C/C++ 要提高效率十倍。需要指明的是这里所说的“窗口”是指 Tcl 定义的窗口，与 X-Windows 与 MS Windows 的定义有所不同，但它可完美地运行在以上两个系统上。

## 二. GID 中二次开发环境的介绍

在 GID 提供了直接利用 TCL/TK 开发界面，命令等功能。具体的有关说明文档可以参见 GID 帮助手册中 TCL-TK Extension 一章中的说明。

下面我们结合创建一个随机四边形的例题来熟悉其开发环境。

```
Proc initGIDProject {dir} {
.....
}

proc userprocess {} {
.....
}
```

以上是 GID 中利用 TCL\_Tk 开发的一般格式。

```
proc InitGIDProject {dir} {

    set materials [.central.s info materials]
    set conditions [.central.s info conditions ovpnt]

    CreateWindow $dir $materials $conditions

}
.....
上面是直接利用 GID 提供的 TCI 命令获得一些 GID 启动环境信息
.....
proc CreateWindow {dir mat cond} {

    set w .gid.win_example

    InitWindow $w "显示工作环境" ExampleCMAS "" \
        "" 1

    frame $w.top
    label $w.top.title_text \
        -text " 创建任意四边形对话框 " -font "songti" -height 2

    frame $w.information -relief ridge -bd 2
```

```

label $w.information.path \
    -text " 程序目录 : $dir " -font "songti" -height 2

label $w.information.conditions \
    -text " 载荷类型 : $cond" -font "songti" -height 2

frame $w.bottom
button $w.bottom.start \
    -text "确定" \
    -height 1 -width 14 -command "CreateRandomSurface $w"
button $w.bottom.random \
    -text "取消" \
    -height 1 -width 20 -command "destroy $w"

pack $w.top.title_text -pady 10
pack $w.information.path \
    $w.information.conditions -side top -anchor w

pack $w.bottom.start $w.bottom.random \
    -side left -anchor center
pack $w.top
pack $w.information -expand yes -fill both
pack $w.bottom -side top -padx 6 -pady 10 -ipady 2
}

```

上面是创建一个对话框，显示 **GID** 的启动目录，现有问题类型的载荷类型以及两个按钮。具体显示形式如下图 1-1 所示。

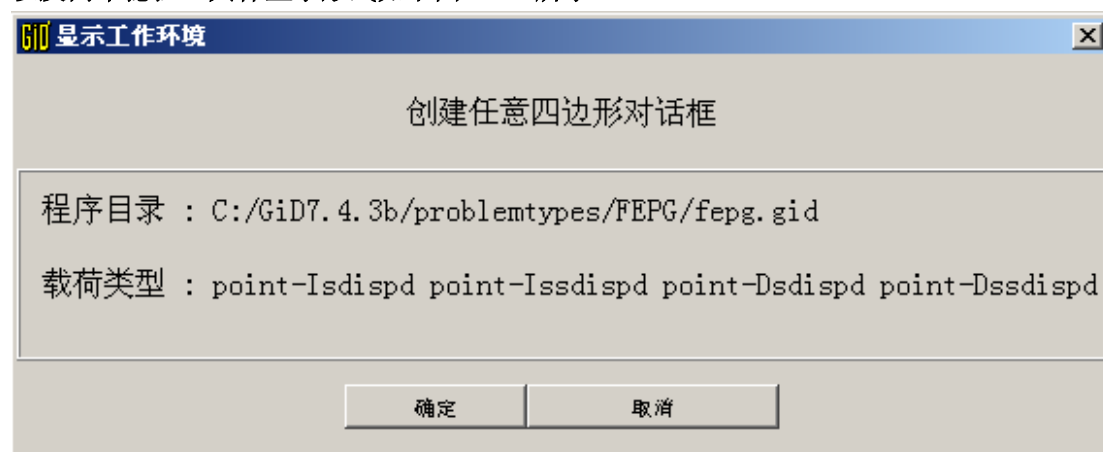


图 1-1 工作环境对话框

```

proc CreateRandomSurface {w} {

set ret [tk_dialogRAM $w.dialog "警告!!" \

```

"提示: 将在图形区域随机产生一个四边形" "" 1 "ok" "cancel"]

```

if {$ret ==0} {
    Create_surface
    destroy $w
}
}

```

上面是创建一个随机的四边形, 在用户确认命令前, 系统给出一个警告提示对话框 (ret 函数定义)。显示效果如下图 1-2 所示。



图 1-2 警告提示对话框

```

proc Create_surface {} {

    set a_x [expr rand()*10]
    set a_y [expr rand()*10]

    set b_x [expr $a_x + rand()*10]
    set b_y [expr $a_y + rand()*10]

    set c_x [expr $b_x + rand()*10]
    set c_y [expr $b_y - rand()*10]

    if {$a_y < $c_y} {
        set d_y [expr $a_y - rand()*10]
        set d_x [expr $a_x + rand()*10]
    } else {
        set d_y [expr $c_y - rand()*10]
        set d_x [expr $c_x - rand()*10]
    }

    .central.s process escape escape escape escape
    .central.s process geometry create line \
    $a_x,$a_y,0.000000to10.176991 \
    $b_x,$b_y,0.000000to10.176991 \
    $c_x,$c_y,0.000000to10.176991 \
    $d_x,$d_y,0.000000to10.176991 \
}

```

close

```
.central.s process escape escape escape escape
.central.s process geometry create NurbsSurface Automatic \
4 \
escape \

.central.s process zoom frame escape escape escape escape
}
```

上面是创建一个随机的四边形进程的定义，在该进程中，使用了大量 GID 提供的命令参数，这些命令参数可以象 TCL 命令一样直接使用。

如

```
.central.s process escape escape escape escape
.central.s process geometry create line \
```

等同于直接在 GID 中创建线段的功能。

其表述方式有写类似 GID 的宏命令功能。关于 GID 的宏命令功能可以参照下面两节说明。

### 三. GID 宏命令文件获得

在 GID 中用户可以实时的保存自己创建模型的宏命令，其实现方法如下：

1. 启动 FEPG.GID
2. 通过主菜单 Utilities>Preferences, 如图 1-3。弹出 Prefences 对话框。
3. 找到 Write batch file 项，选中，并点击右侧浏览对话框扭，选择你需要保存批命令的路径，以及文件名。如图 1-4 所示中红色框图。

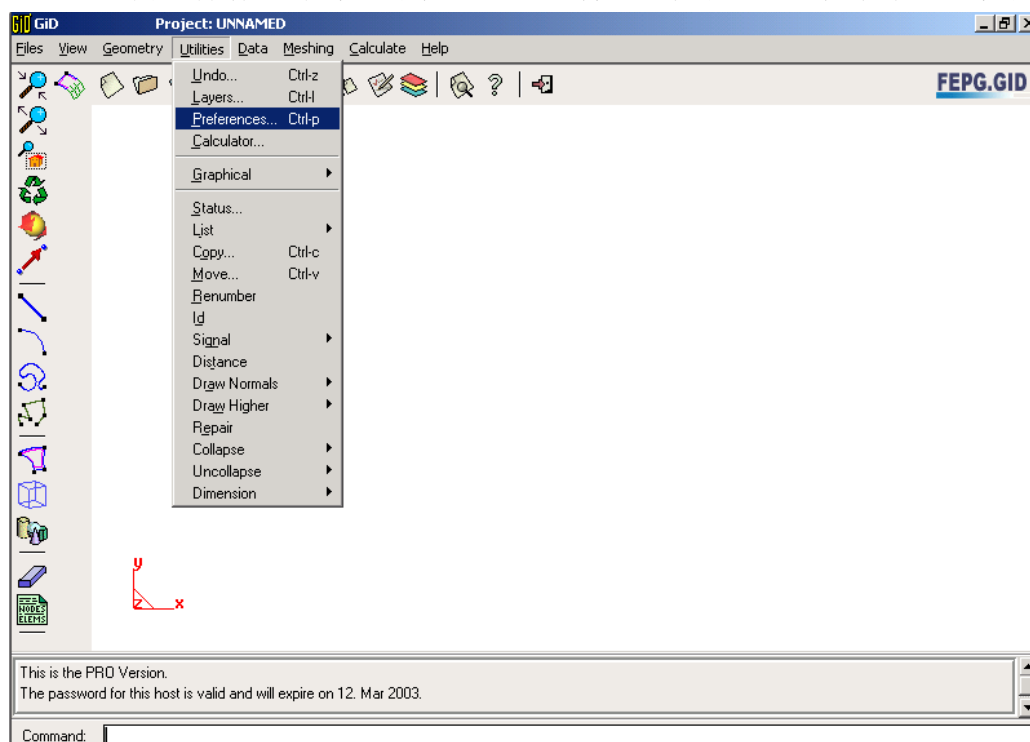
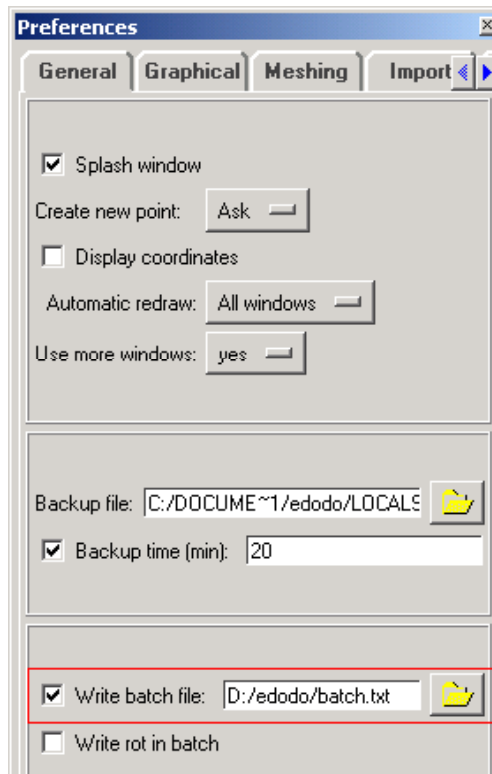


图 1-3 Preferences 菜单



1-4Preferences 对话框。

#### 四. 打发打发

在 GID 中可以在 Command 命令中直接输入命令的方式来创建模型。

例如创建一条直线的宏命令的格式如下：

依次在 Command 命令行中输入：

geometry

create

line

0 0

1 0

这就创建除了一条以 0, 0 和 1, 0 为坐标的直线。

在 GID 中，宏命令的格式是按照右侧菜单的摁键名称来规定的，用户可以通过如下方式显示 GID 右侧菜单：

1. 选择 GID 菜单 Utilities>Graphical>Toolbars 弹出下图 1-5 所示
2. 弹出 Toolbars 对话框，选择 Right buttons 选项，选中 inside 单选框。如图 1-6 所示。
3. 则 GID 图形区域右侧就会出现如图 1-7 所示的右侧菜单。

说明：用户每次在 Command 命令行中输入一个右侧菜单的名称，右侧菜单就会自动进入下一级菜单。用户根据右侧菜单的变化就可以找到相应的命令对应的宏命令名称。

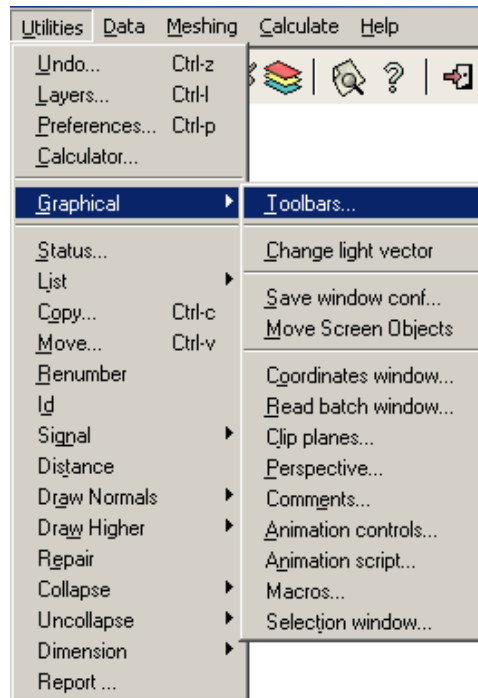


图 1-5 Toolbars 菜单

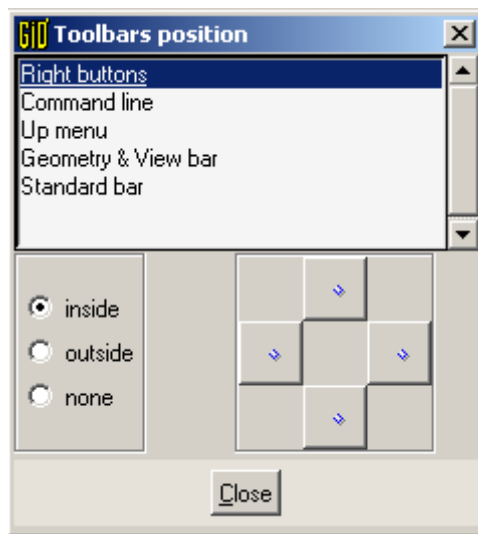


图 1-6 Toolbars 对话框

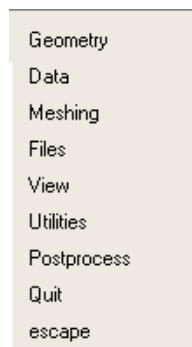


图 1-7 右侧菜单

## 五. 参数化模型的实现

根据前面四节的介绍, 大体了解了 GID 对于 TCL\_Tk 语言的支持, 以及宏命令语言的格式, 我们就可以利用这些来实现参数化模型的实施。

首先, 在 GID 中, 利用 TCL\_TK 语言编写界面, 用以输入变量参数的值, 注意编写的 TCL 文件需要存放在用户编写的 GID 安装目录下面的 fepg.gid 目录中。

接着, 我们可以利用 TCL\_TK 语言的格式, 来直接在 GID 软件内部实现模型的创立。例如前面给出的例题。

最后, 我们启动 GID, 在选择问题类型的时候, 选择 FEPG, 此时用户界面就会自动弹出。如图 1-1 的用户界面。

或者我们也可以这样来作, 利用现有的其他编程语言, 比如 VC/VB/Java/perl 等, 编写用户输入参数界面, 然后根据用户输入参数, 直接写出模型的宏命令文件, 然后用 GID 的阿模型导入功能实现模型的输入。

### 参考文献:

《GID Reference Manual》 CIMNE (International Center for Numerical Methods in Engineering)

《GID User Manual》 CIMNE (International Center for Numerical Methods in Engineering)

《Tcl/TK 组合教程》 Brent B. Welch 著 王道义 乔陶鹏 等译 电子工业出版社

《Tcl/Tk 编程权威指南》 Brent B. Welch 著 崔凯 译 中国电力出版社