致尊敬的顾客

关于产品目录等资料中的旧公司名称

NEC电子公司与株式会社瑞萨科技于2010年4月1日进行业务整合（合并），整合后的新公司暨“瑞萨电子公司”继承两家公司的所有业务。因此，本资料中虽还保留有旧公司名称等标识，但是并不妨碍本资料的有效性，敬请谅解。

瑞萨电子公司网址：http://www.renesas.com

2010年4月1日
瑞萨电子公司

【发行】瑞萨电子公司（http://www.renesas.com）

【业务咨询】http://www.renesas.com/inquiry

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

HEW Tcl/Tk 应用笔记

HEW Tcl/Tk
应用笔记

目录

HEW 中的 Tcl/Tk 概述

Tcl/Tk 是 HEW（高性能嵌入式工作区）中支持的脚本语言。目标版本是 Tcl/Tk 版本 8.4.1。

Tcl/Tk 包含脚本语言的类成员“Tcl”（工具命令语言）和“Tk”（工具套件），用于编程图形用户界面。脚本语言 Tcl/Tk 不需要进行编译，程序的执行结果也会立即反映出来。

Tcl 具有能够尽可能简化编程的语法。Tcl 可用于独立应用程序，也可以内建至应用程序中。Tk 使它可以构建能够恰当符合用户需要的 GUI。

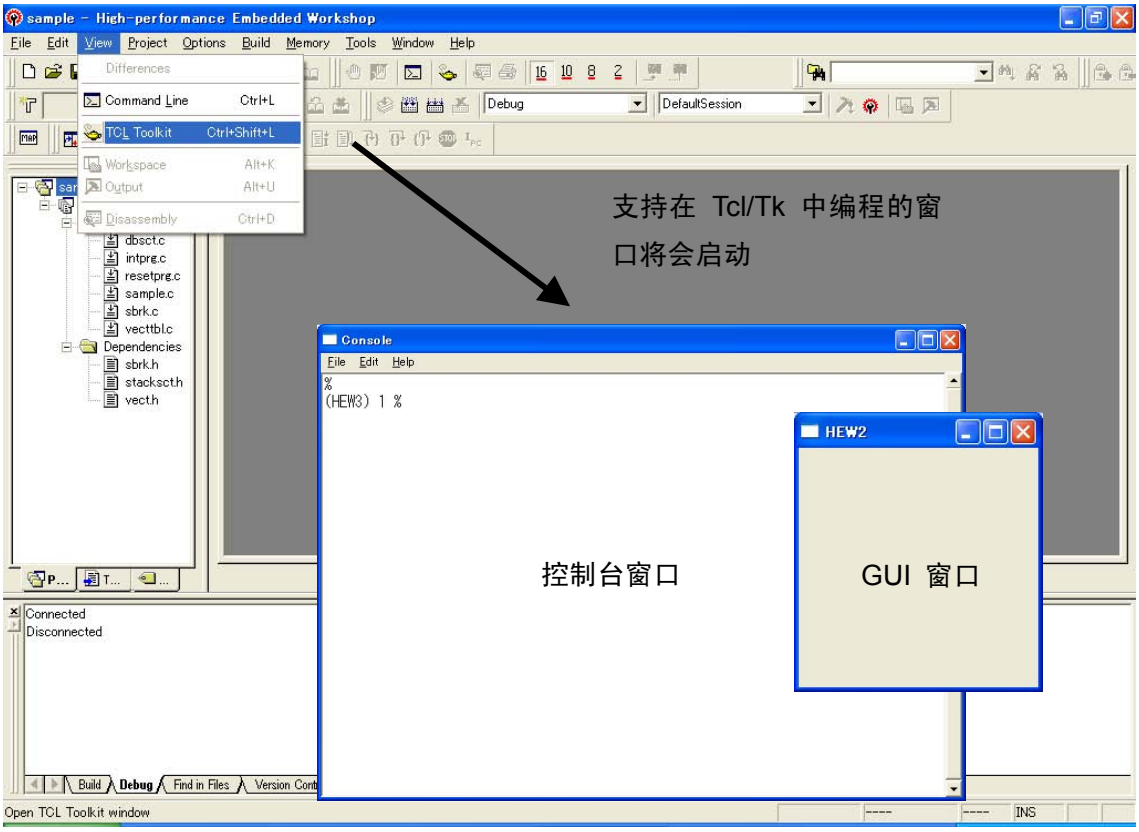
HEW 支持 Tcl/Tk。可使用默认情况下 HEW 中所预备的函数和 GUI，也可通过在 Tcl/Tk 中编程来自定义 GUI 环境以满足用户的个别需要。

1，使用 Tcl/Tk

1-1，启动 Tcl/Tk

本章将描述如何使用 HEW 中的 Tcl/Tk 命令。

从 HEW 命令菜单选取“视图”(View) – “TCL 工具套件”(TCL Toolkit)。支持在 Tcl/Tk 中编程的控制台窗口和 GUI 窗口将会启动。您可以使用这些窗口在 Tcl/Tk 中编程。



图：启动 Tcl/Tk

控制台窗口

您可以在 Tcl/Tk 中编程然后使用解释程序在控制台窗口上执行它。您也可以加载预先编写的脚本文件和执行它们。

GUI 窗口

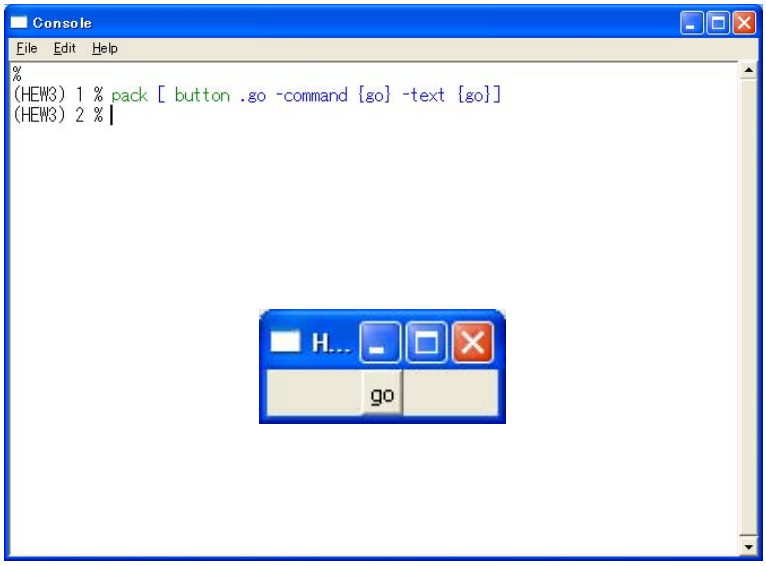
控制台窗口上程序的执行结果将会在 GUI 窗口上反映。默认情况下，上层窗口在 HEW 中预备。此外，除了上层窗口，您也可以建立新的 GUI 窗口。

1-2，Tcl/Tk 的执行方法

您可以按照下列 3 种方法执行 HEW 中的 Tcl/Tk。

① 解释程序方法

在解释程序方法中编程时，您可以在控制台窗口上分配 Tcl/Tk 的命令。所分配的 Tk 命令将会在启动 Tcl/Tk 工具套件时反映于 GUI 窗口上。

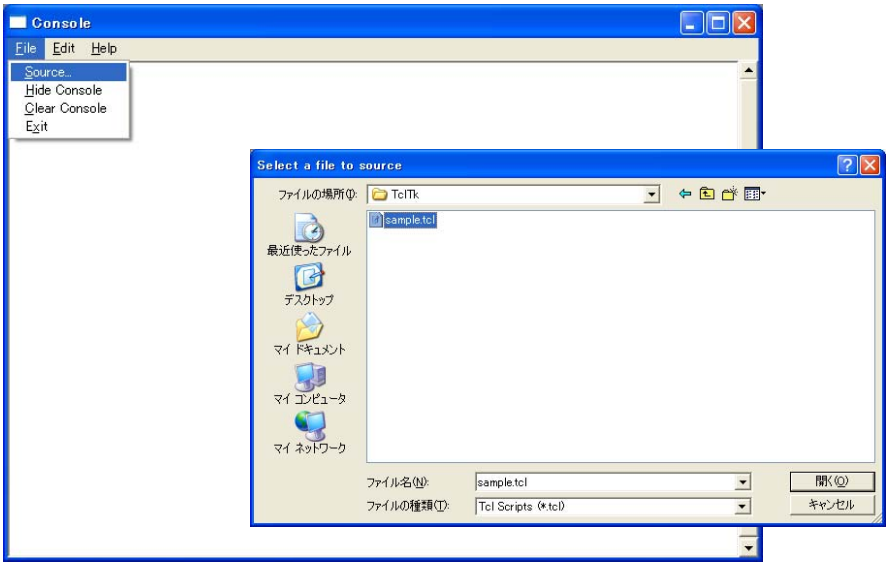


例如：把 HEW 的程序执行命令“go”分配到一个按钮上。

图：在解释程序方法中编程

② 执行加载的脚本文件

您可以预先使用脚本文件在 Tcl/Tk 中编程，然后将该文件加载到控制台窗口上。从控制台窗口的命令菜单选取“文件”(file) –“源”(Source)，“选取文件至源”(Select a file to source) 窗口将会打开。窗口打开之后，指定预先建立的 Tcl/Tk 脚本文件。



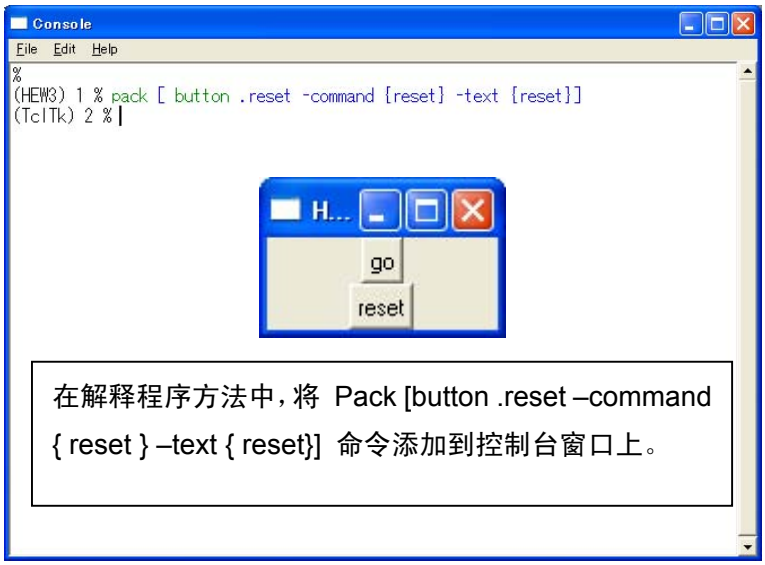
图：选择 Tcl/Tk 源文件

图：选择源文件之后

选择源文件之后，脚本文件中的编程将会反映在控制台窗口上。

③ 同时使用解释程序方法和脚本加载方法执行

加载预先建立的脚本文件之后，您可以使用解释程序方法继续添加脚本。



例如：添加 HEW“reset”的复位命令

图：程序添加

2，Tcl/Tk 的基本编程方法

2-1，Tcl 的编程方法

2-1-1，基本语法

Tcl 具有非常简单的语法。Tcl 的内建命令和用户建立的 procedure 将会作为 Tcl 编程的命令使用。您可以使用一个空格将命令所需的命令和参数隔开，然后构建一个执行格式。您也可以另起一行将它们隔开，或使用分号来描述复杂的执行格式。

基本语法 1：使用空格隔开来列出参数。

```
Command arg1 arg2 arg3 …
```

基本语法 2：在一行中列出数个命令的情形下使用分号 (;)。

```
Command arg1 arg2 arg3 …; Command arg1 arg2 arg3 …
```

基本语法 3：在数行中列出一个命令的情形下使用 \ 符号。

```
Command arg1 \
        arg2 arg3…
```

基本语法 4：描述注解的情形下使用 # 符号。

```
Command arg1            #comments
```

Tcl 的内建命令和用户建立的 procedure 将作为命令使用。

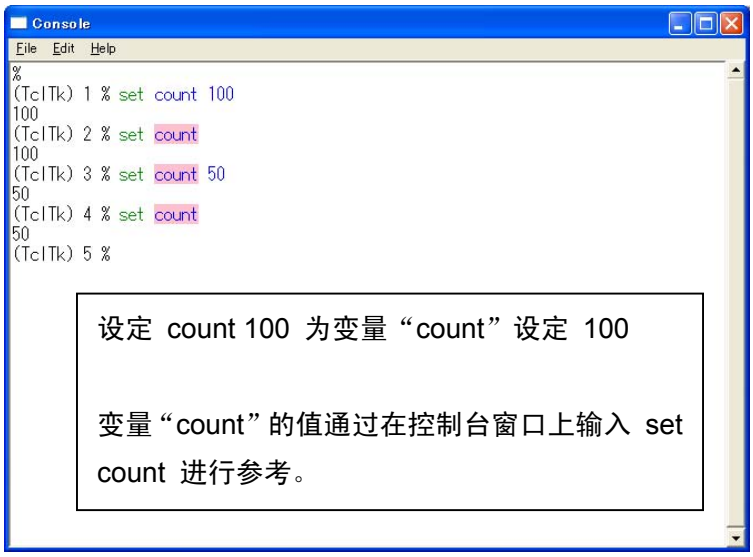
Arg 是参数，是 Tcl 的内建命令和用户建立的 procedure 所需要的。

2-2，Tcl 的内建命令

本章将描述 Tcl 中预备的内建命令。本章所介绍的内建命令为编程的最低需要。

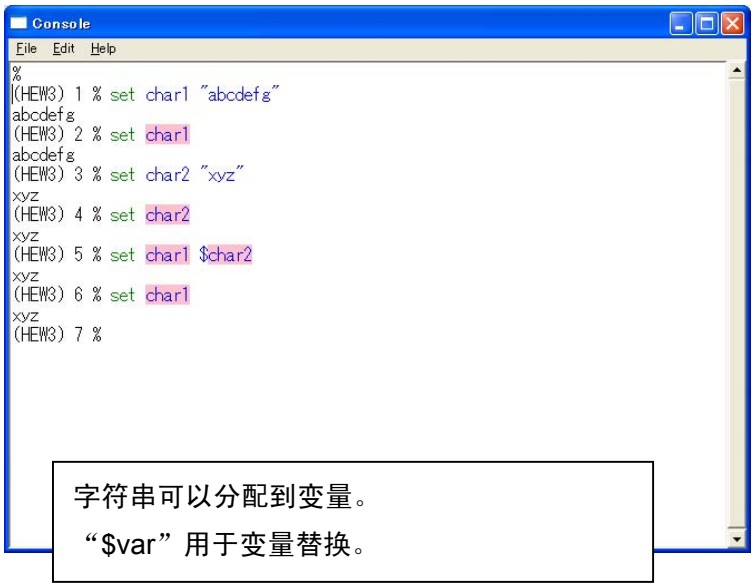
2-2-1，变量

Tcl 中的变量不需要在使用前声明类型。您可以按照需要来命名它们。您也可以对变量设定一个值，并在 Tcl 编程中参考变量的值。



图：变量的设定和参考

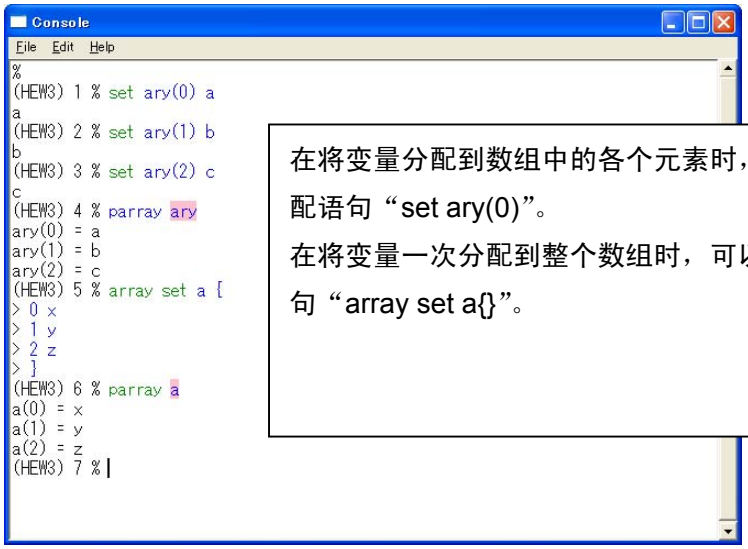
您也可以按照需要在 Tcl 中设定变量的字符串



图：分配字符串给变量

2-2-2，数组

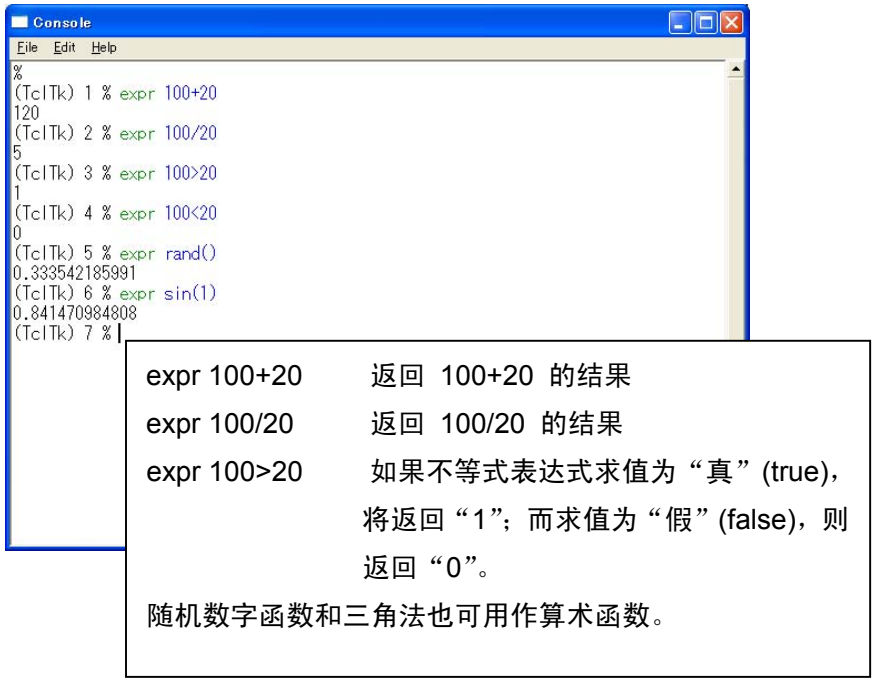
数组可以在 Tcl 中使用。您可以将变量分配给数组中的各个元素，也可以一次分配给整个数组。



图：数组

2-2-3，算术操作

整数操作、浮点操作和不等式的比较可使用 expr 命令在 Tcl 中执行。运算符和数学函数将会在 Tcl 中预备为内建命令。



图：算术操作

下表为您显示 Tcl 中支持的运算符和函数，以及它们在 Tcl 中的含义。

列表：运算符

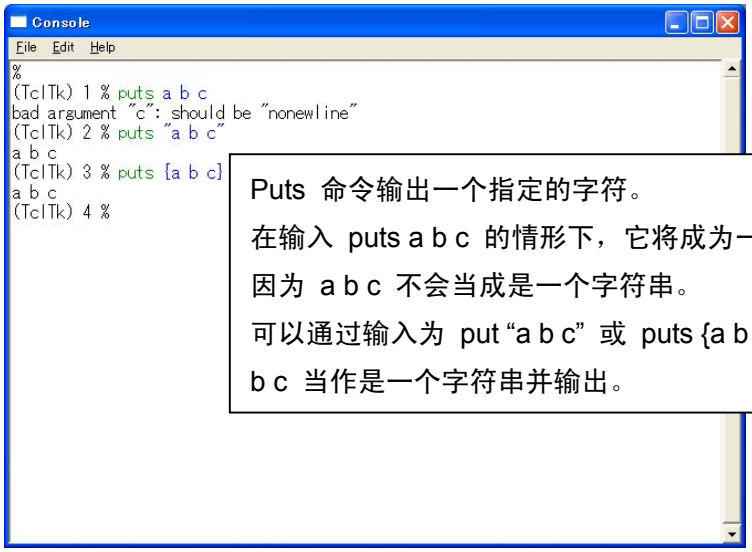
符号	含义
-, +, ~ ,!	减号，加号，补数，反
*, /, %	乘，除，余数
+, -	加，减
<<, >>	左移，右移
<, >	布尔表达式中的比较 （大于，小于）
<=, >=	布尔表达式中的比较 （大于等于，小于等于）
==, !=	等号，布尔表达式中的不等式符号
Eq, ne	等式，布尔表达式中的不等式（在字符串中使用）
&, ^	比特级“与”(AND)，比特级“异或”(XOR)
&&, ||	逻辑“与”，逻辑“或”
x?y:z	条件

列表：函数

函数	含义
Acos, cos, hypot, sinh, asin(), cosh(), log(), sqrt(), atan(), exp(), log10(), tan(), atan2(), floor(), pow(), tanh(), ceil(), fmod(), sin()	数学函数
abs(arg)	绝对值
double(arg)	双精度值
int(arg)	整数值
Rand()	随机数值
round(arg)	舍入为整数的值
srand(arg)	随机数字的初始值

2-2-4，双引号和花括号

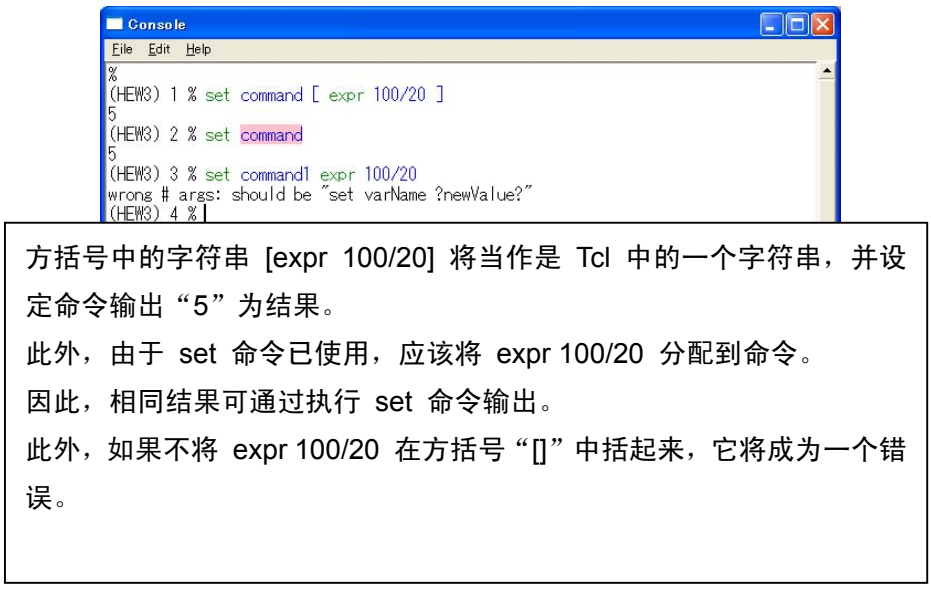
双引号 ”“ 或花括号 {} 中的字符串将当作 Tcl 中的一个字符串。



图：双引号和花括号

2-2-5，命令的分配

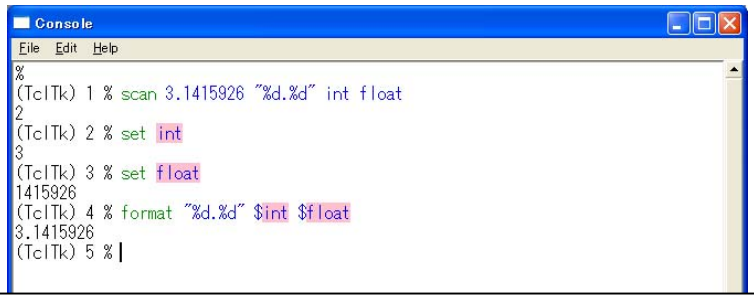
方括号 [] 中的字符串将当作 Tcl 中的一个命令。



图：命令的分配

2-2-6，输入-输出格式

scan 命令和 format 命令将会在 Tcl 中预备为 I/O 命令。这些命令相等于 ANSI 中的 scanf 和 printf。此外，您还可以使用“%”来定义格式的样式。

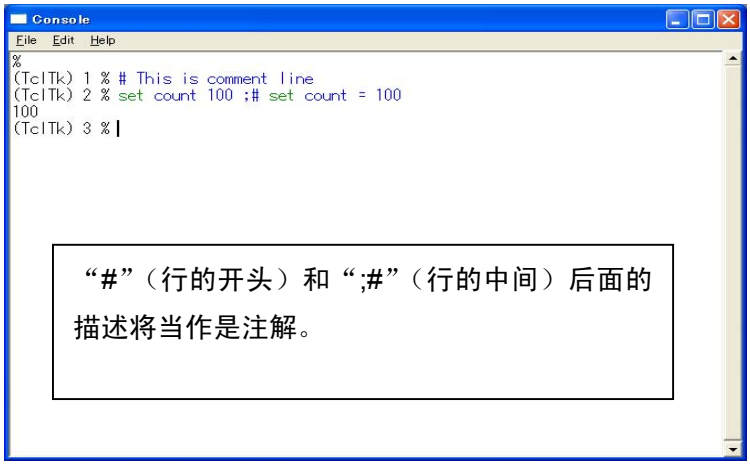


scan 3.1415926 “%d.%d” int float 相等于 ANSI 中的 scanf。
Scan 命令会将整个部分“3”和 3.1415926 的分数部分“1415926”分别扫描到 int 和 float 变量。此外，它将会返回 2（输入值的次数）作为执行的结果。
format “%d.%d” $int $float 相等于 ANSI 中的 printf。
该命令返回 3.1415926 作为执行的结果。

图：输入-输出格式

2-2-7，注解的描述

如果要在 Tcl 的脚本文件中描述注解，请将“#”添加到行的开头。
此外，要在行的中间插入注解，则请将“;#”添加到注解的开头。



“#”（行的开头）和“;#”（行的中间）后面的描述将当作是注解。

图：注解的描述

2-2-8，Procedure

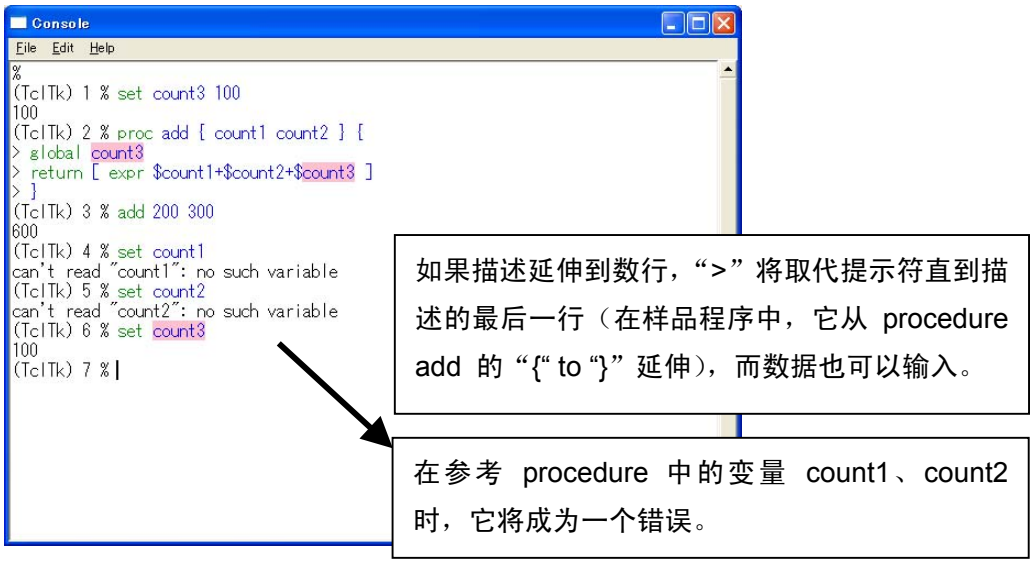
在 Tcl 中，一行命令可当作 C 语言中的一个函数。称为“procedure”的函数可在 Tcl 中作为内建命令使用。

可将 procedure 建立为一个函数，并通过使用 proc 命令取得 0 或更多参数。在 procedure 中所定义的变量将当作本地变量，并且只可以在该 procedure 中参考。此外，如果要在该 procedure 中参考全局变量，则需要在 procedure 中定义全局变量。

如何建立“procedure”的实例

```
~样品程序~
set count3 100                  ;#将 100 分配到变量 count3 作为外部变量
proc add { count1 count2 }{     ;#接受参数 count1 和 count2
        global count3           ;#参考 count3 时需要全局定义
        return[ expr $count1+$count2+$count3]
                                #定义由 procedure add 返回的值
                                #使用 return 命令
}
add 200 300                     ;#使用 procedure add
```

编写样品程序时，procedure add 将会调用。而“600”将会返回为“add 200 300”的结果。



图：procedure 的建立

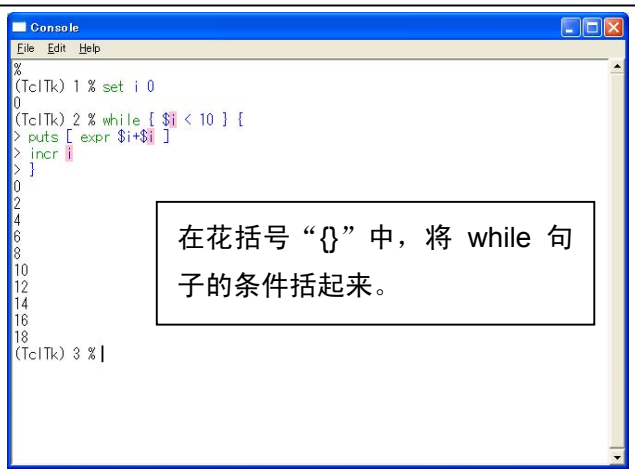
2-2-9，控制结构命令

Tcl 支持其他高级语言中存在的实质控制结构。

控制结构命令包括 while、for、if、if…else、switch、foreach 等等。

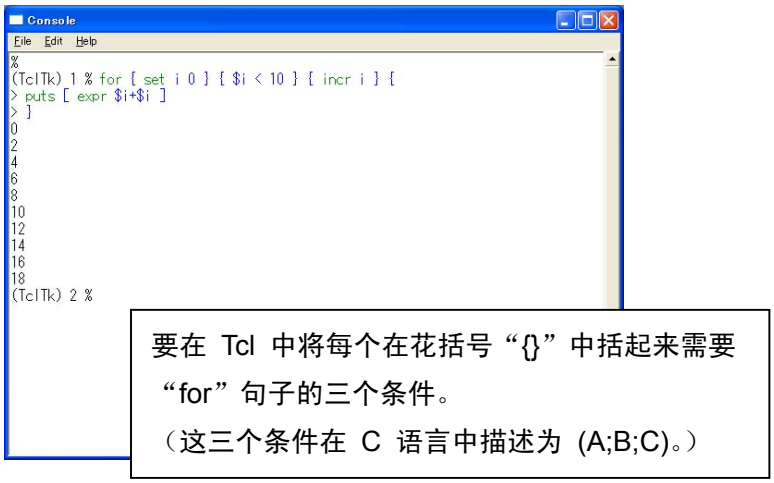
① 如何使用“while”的实例

```
set i 0
while { $i < 10 } {
        puts [ expr $i+$i ]
        incr i
}
```



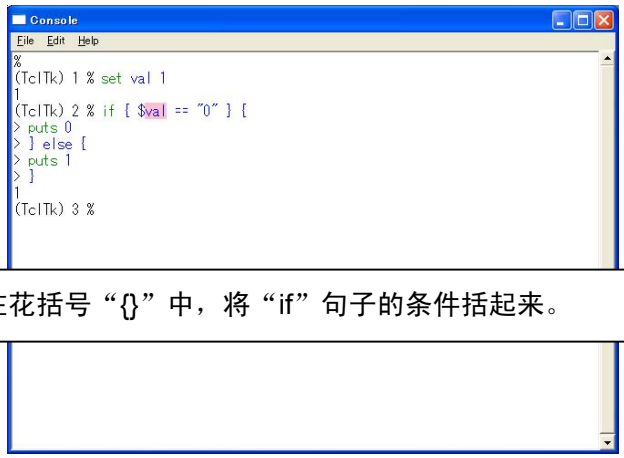
② 如何使用“for”的实例

```
for { set i 0 } { $i < 10 } { incr i } {
        puts [ expr $i+$i ]
}
```



③ 如何使用“if…else”的实例

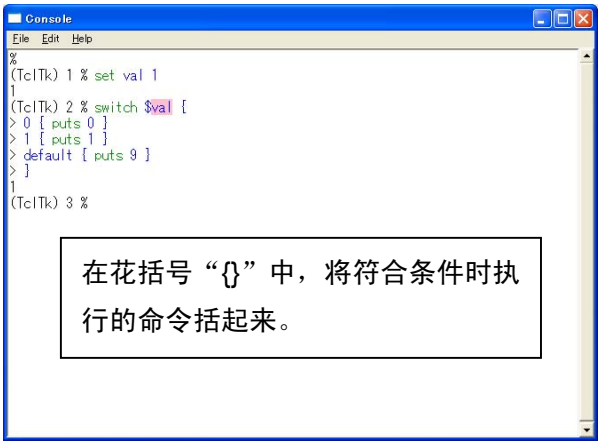
```
set val 1
if { $val == “0” } {
        puts 0
} else {
        puts 1
}
```



在花括号“{}”中，将“if”句子的条件括起来。

④ 如何使用“switch”的实例

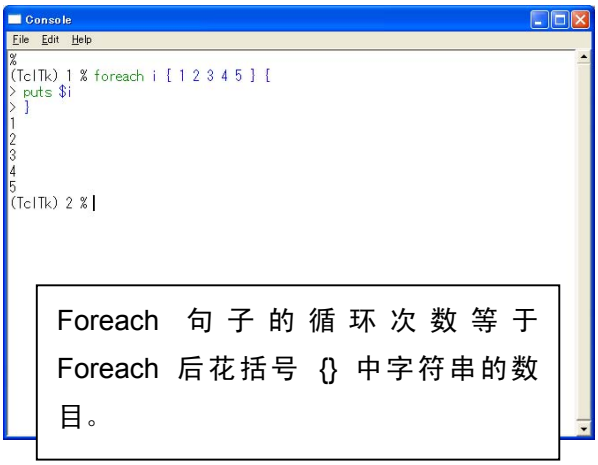
```
set val 1
switch $val {
        0 { puts 0 }
        1 { puts 1 }
        default { put 9 }
}
```



在花括号“{}”中，将符合条件时执行的命令括起来。

⑤ 如何使用“foreach”的实例

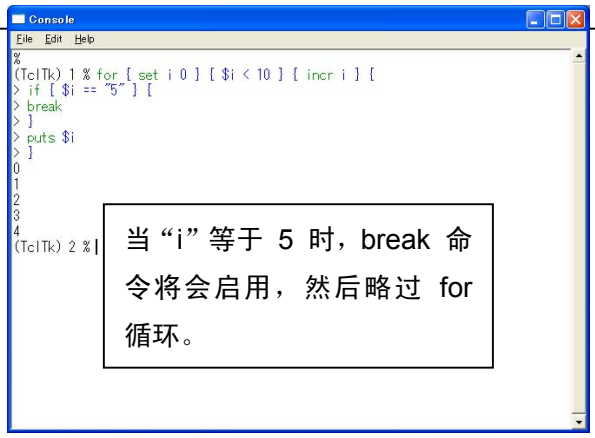
```
foreach i { 1 2 3 4 5 } {
        puts $i
}
```



Foreach 句子的循环次数等于 Foreach 后花括号 {} 中字符串的数目。

⑥ continue 和 break 可在 while、for 和 foreach 句子中使用。

```
for { set i 0 } { $i < 10 } { incr i } {
        if { $i == “5” }{
                break                ;#Break 可使用
        }
        puts $i
}
```



当“i”等于 5 时，break 命令将会启用，然后略过 for 循环。

2-3，Tk 的编程方法

2-3-1，基本语法

窗口小部件（Tk 中支持的 GUI 组件）在 GUI 窗口上定义和分配，它由用户在 Tk 编程中新建立。

窗口小部件的定义必须根据 Tk 编程的语法规则使用白空格来隔开“widget”、“path”和“option”进行描述。

```
widget .path –option1 –option2 –option3 …
```

path 名称必须以句点“.”开头。“.path”命令在执行 widget 命令之后生成。您可以在新建立的 GUI 窗口上分配“.path”命令。

```
pack .path
```

可以使用用于分配“.path”命令的 pack 命令在 GUI 窗口上分配“.path”命令。

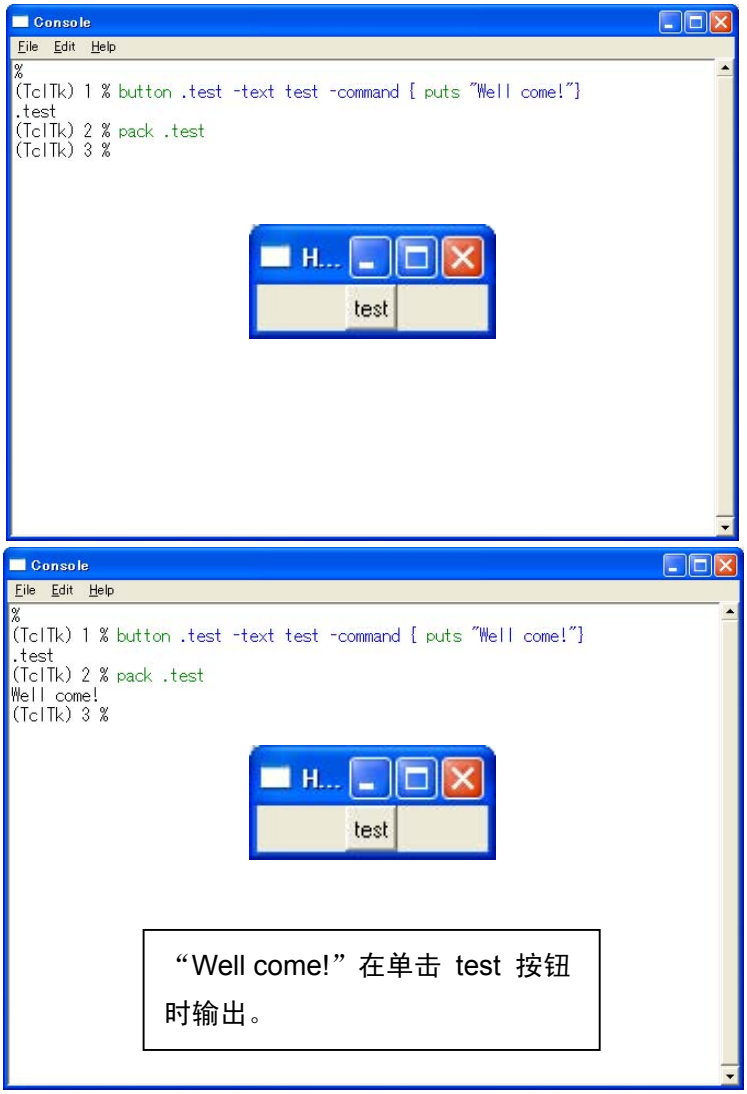
可以建立各种环境来满足用户在 Tk 编程中的特殊需要。

2-4，Tk 中的窗口小部件

2-4-1，建立按钮

按钮可以使用 Tk 的 button 命令建立。

```
button .test –text test –command {puts “Well come!”}
pack .test
```



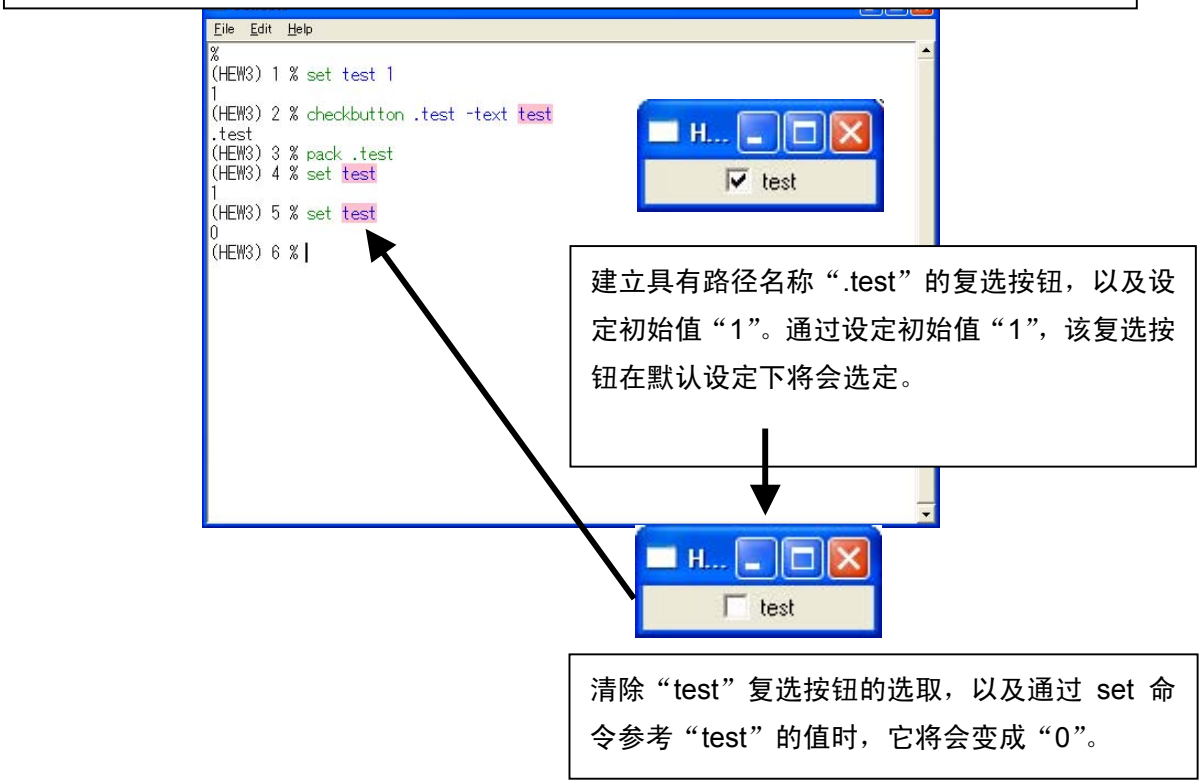
图：按钮的建立

在上述样品程序中，建立名为“test”的按钮。按钮的名称可以在 -text 选项中指定。单击按钮时所执行的命令可在 -command 选项中指定。所以，在单击“test”按钮时，花括号“{}”中的命令将会执行，并且在控制台窗口上输出“Well come!”。

2-4-2，建立复选按钮

复选按钮可以使用 Tk 的 checkbutton 命令建立。

```
set test 1                              ;#将“1”设定到变量“test”作为初始值。
checkbutton .test –text test
pack .test
```



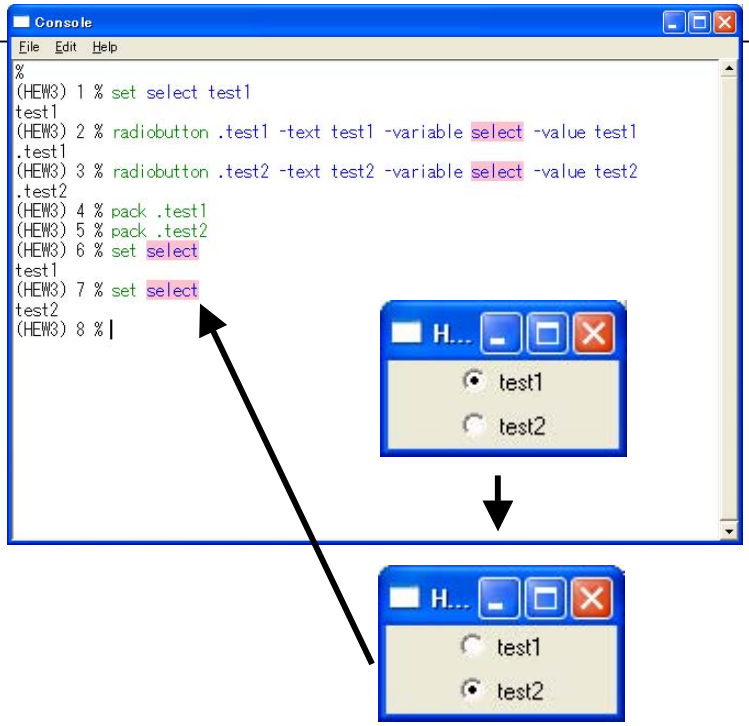
图：复选按钮的建立

在上述样品程序中，建立名为“test”的按钮。按钮的名称可以在 -text 选项中指定。
此外，“1”的初始值可通过使用 set 命令设定为变量“test”。

2-4-3，建立单选按钮

单选按钮可以使用 Tk 的 radiobutton 命令建立。

```
set select test1                ;#在默认设定下，test 1 的复选按钮将会选定。
radiobutton .test1 –text test1 –variable select –value test1
radiobutton .test2 –text test2 –variable select –value test2

pack .test1
pack .test2
```



单击 test2 的单选按钮时，test2 将会替换为在 –variable 选项中选定的值。

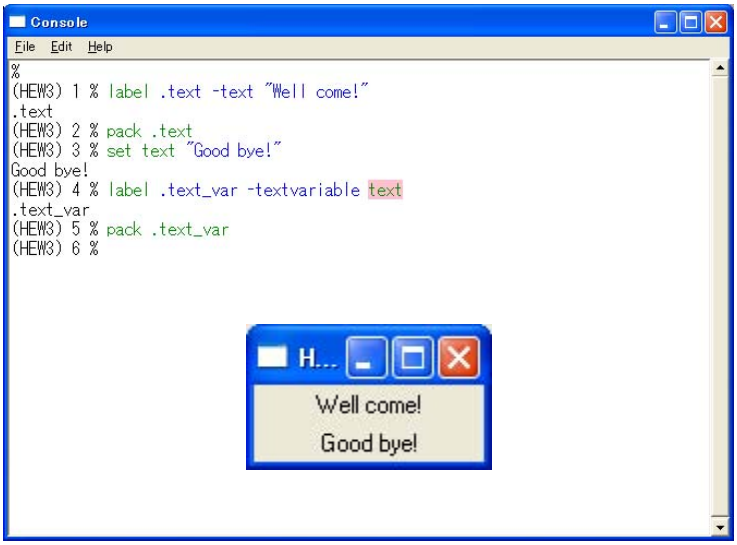
图：单选按钮的建立

选定按钮的 -value 将会传递到在 –variable 选项中选取的变量。在上述样品程序中，由于在建立单选按钮时将“test1”选定为变量，因此单选按钮“test1”将会在默认设定下被选定。此外，通过将 –variable 选项中指定的多个单选按钮的变量命名为相同名称，这些复选框将只能唯一选定。

2-4-4，建立标签

单行信息可以使用 Tk 中的 label 命令在 GUI 窗口上显示。

```
label .text –text “Well come!”
pack .text

set text “Good bye!”
label .text_var –textvariable text
pack .text_var
```



图：标签的建立

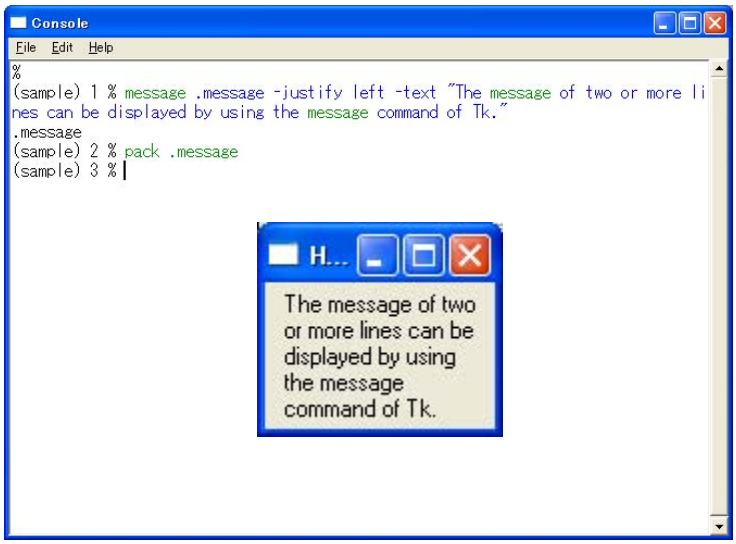
在 label 命令的 -text 选项中选定的字符串将会在 GUI 窗口上显示。

此外，也可显示预先使用 –textvariable 选项设定为变量“test”的字符串。

2-4-5，建立信息

数行信息可以使用 Tk 中的 message 命令在 GUI 窗口上显示。

```
message .message –justify left –text “The message of two or more lines can be displayed by using the message command of Tk.”
pack .message
```



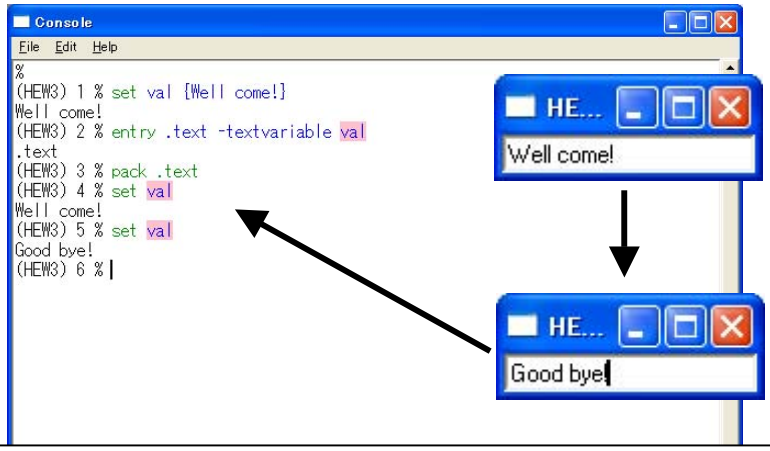
图：信息的建立

虽然使用 label 命令仅可显示单行信息，但数行信息可以通过使用 message 命令显示。此外，如果命令信息太长而无法在一行中输入，您可以使用“\”开始新的一行。

2-4-6，建立输入框

输入对话框可以使用 Tk 的 entry 命令建立。

```
set val {Well come!}
entry .text –textvariable val
pack .text
```



“Well come!”通过使用 set 命令设定为变量 val 的初始值。因此，在建立输入对话框时，“Well come!”将会在默认情况下显示。在输入对话框中输入“Good bye!”时，输入字符串“Good bye!”将会反映在变量 val 中。

图：输入框的建立

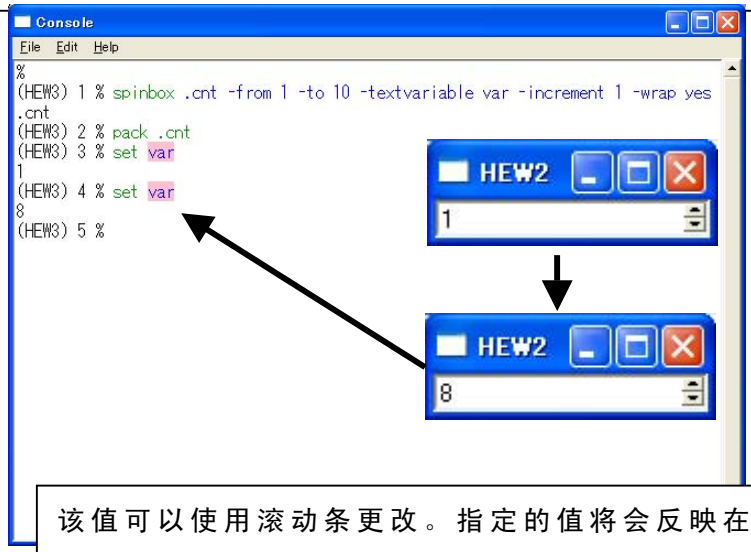
使用 entry 命令建立的输入对话框只可以接受单行字符串。输入字符串将会反映在 –textvariable 选项中指定的变量 val 中。

如果要建立可接受超过一个单行字符串的输入对话框，您可以使用 Tk 中预备的 text 命令。此外，text 命令也为您提供多种选项。通过使用这些选项，您也可以建立简单的编辑程序。

2-4-7，建立旋转框

具有滚动条的旋转框可以使用 Tk 的 spinbox 命令建立。

```
spinbox .cnt –from 1 –to 10 –textvariable var –increment 1 –wrap yes
pack .cnt
```



该值可以使用滚动条更改。指定的值将会反映在 –textvariable 中指定的变量 val 中。

图：旋转框的建立

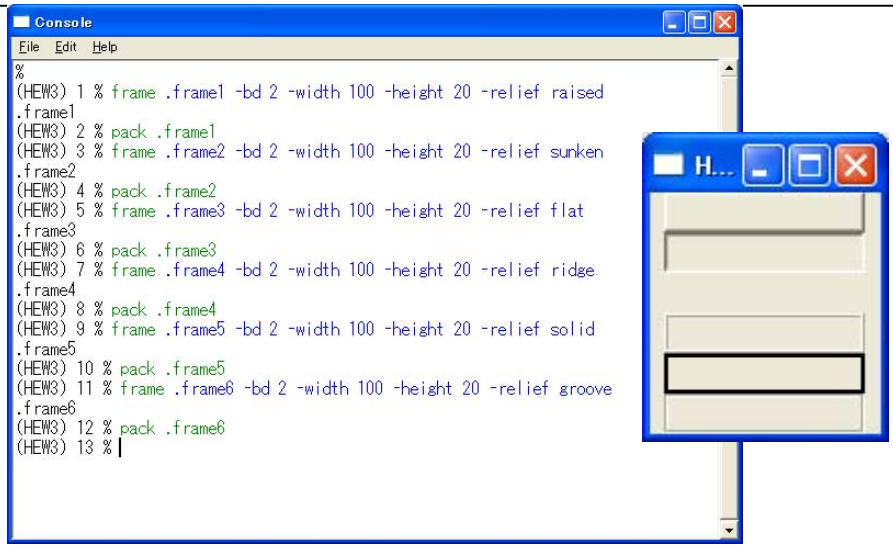
您可以使用 spinbox 命令的 –from 和 –to 选项指定旋转框中可设定值的范围。您也可以使用 –increment 选项指定旋转框中显示的值之间的间隔。指定的值将会反映在 –textvariable 中指定的变量 val 中。

此外，如果在 –wrap 选项中选取“是”(yes)，该值会在您于 –from 和 –to 选项中指定的范围之中循环。（例如，0 ~..~ 10 ~ 0 ~..~ 5）

2-4-8，建立框架

框架可以使用 Tk 的 frame 命令建立。

```
frame .frame1 –bd 2 –width 100 –height 20 –relief raised
pack .frame1
frame .frame2 –bd 2 –width 100 –height 20 –relief sunken
pack .frame2
frame .frame3 –bd 2 –width 100 –height 20 –relief flat
pack .frame3
frame .frame4 –bd 2 –width 100 –height 20 –relief ridge
pack .frame4
frame .frame5 –bd2 –width 100 –height 20 –relief solid
pack .frame5
frame .frame6 –bd2 –width 100 –height 20 –relief groove
pack .frame6
```



图：框架的建立

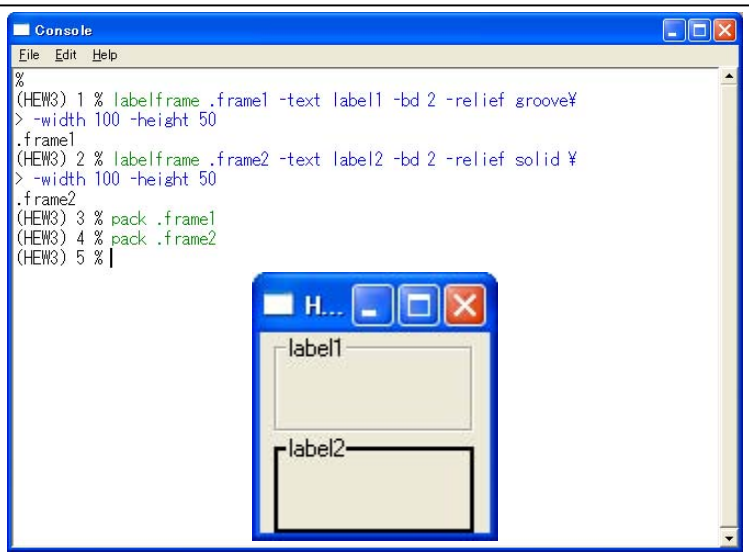
用户新建立的 GUI 窗口可使用 frame 命令排列。将窗口小部件分配到框架的方法将会在稍后章节中描述。

您可以在 frame 命令的 –width 和 –height 选项中指定框架的大小。您也可以在 relief 选项中更改框架的形状。

2-4-9，建立标签框架

标签框架可以使用 Tk 的 labelframe 命令建立。

```
labelframe .frame1 –text label1 –bd 2 –relief groove –width 100 –height 50
labelframe .frame2 –text label2 –bd 2 –relief solid –width 100 –height 50
pack .frame1
pack .frame2
```



图：标签框架的建立

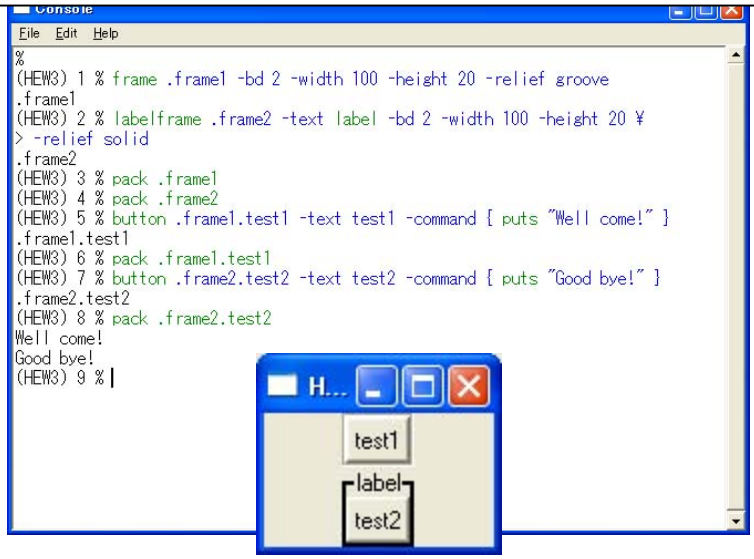
用户新建立的 GUI 窗口可使用 labelframe 命令以及 frame 命令排列。frame 命令的差异在于您可以使用 –text 选项为框架命名。

此外，您可以在 labelframe 命令以及 frame 命令的 –width 和 –height 选项中指定框架的大小。您也可以在 labelframe 命令以及 frame 命令的 relief 选项中指定框架的形状。

2-4-10，将窗口小部件分配到框架/标签框架

您可以将 2-4-9、2-4-10 章节中建立的 Tk 窗口小部件分配到框架/标签框架。

```
frame .frame1 –bd 2 –width 100 –height 20 –relief groove
labelframe .frame2 –text label –bd 2 –width 100 –height 50 –relief solid
pack .frame1
pack .frame2

button .frame1.test1 –text test1 –command { puts “Well come!” }
pack .frame1.test1
button .frame2.test2 –text text2 –command { puts “Good bye!” }
pack .frame2.test2
```



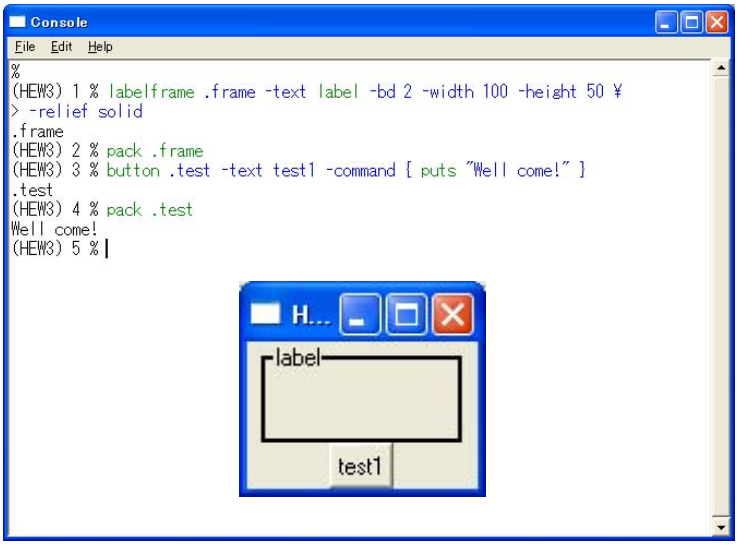
图：分配窗口小部件的实例

在将 Tk 窗口小部件分配到框架或标签框架的情形下，路径名称将会和平常的不一样。

在上述样品程序中，按钮在框架和标签框架上分配。框架的路径名称是“.frame1”，而标签框架的路径名称是“.frame2”。因此，框架和标签框架上所分配之按钮的路径名称各指定为“frame1.test1”和“frame1.test2”。这些路径名称表示按钮“test1”分配到框架“.frame1”，而按钮“test2”分配到标签框架“.frame2”。

此外，如果不使用框架或标签框架的路径名称而是使用 Tk 窗口小部件的路径名称，窗口小部件将会分配到框架或标签框架的外面。

```
labelframe .frame –text label –bd 2 –width 100 –height 50 –relief solid
pack .frame

button .test –text test1 –command { puts “Well come!” }
pack .test
```



图：分配窗口小部件的实例

~补充信息~

Tcl/Tk 路径名称中使用的“.”（句点）显示路由。通过将框架和标签框架分配到路由，GUI 环境中的路径名称将显示分层的系统。

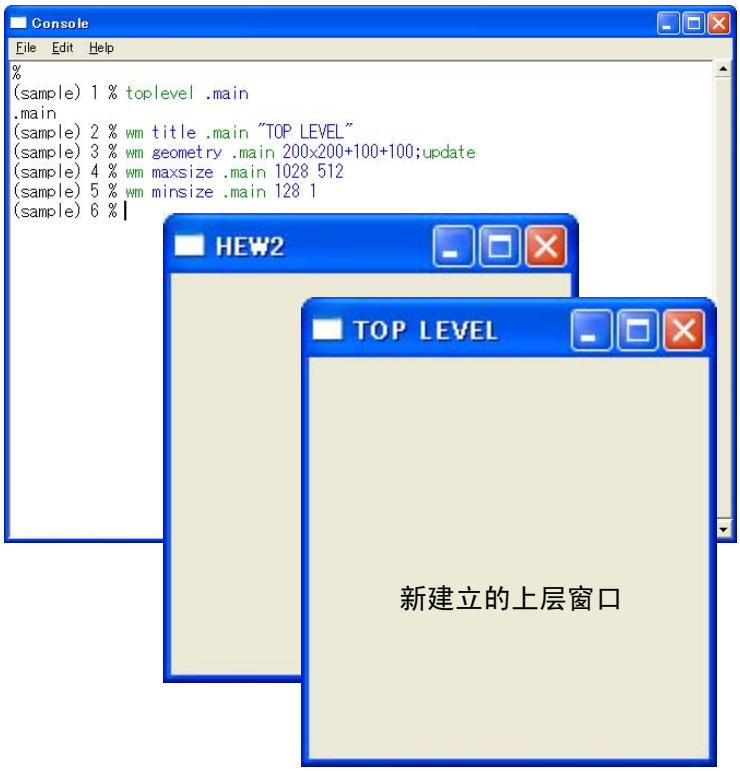
例如：

路径名	.top	GUI 环境的上层
	.top.frame1	分配到上层的 frame1 的路径名
	.top.frame2	分配到上层的 frame2 的路径名
	.top.frame2.subframe	分配到 frame3 的子框架的路径名

2-4-11，建立新的上层窗口

上层窗口可以使用 Tk 的 toplevel 命令新建立。

```
toplevel .main
wm title .main “TOP LEVEL”
wm geometry .main 200x200+100+100; update
wm maxsize .main 1028 512
wm minsize .main 128 1
```



图：上层窗口的建立

在上述样品程序中，除了默认情况下在 HEW 中预备的窗口外，还有使用 toplevel 命令新建立的窗口。新窗口的标题和大小可以使用 wm 命令指定。

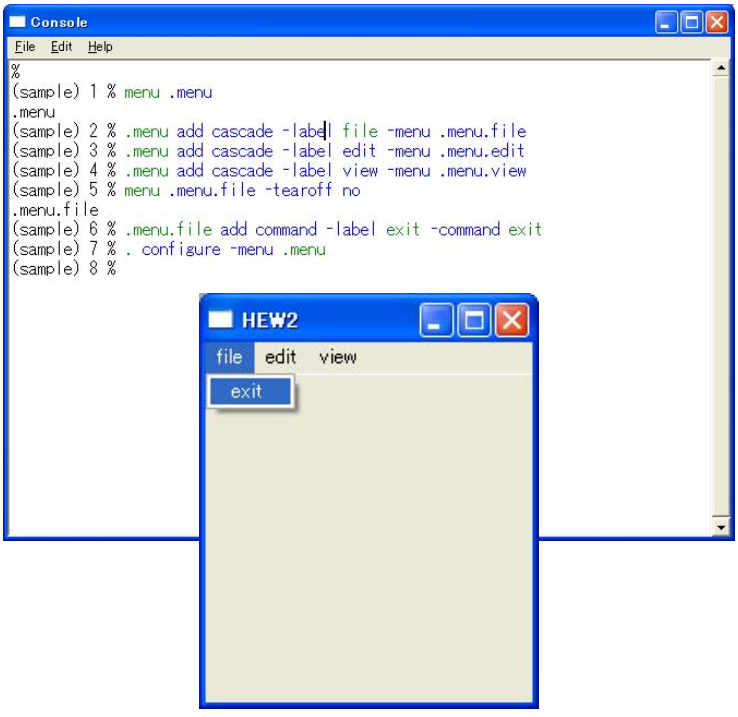
如果要在新窗口上分配 Tk 的窗口小部件，路径名称必须以“.main”开头。

此外，新窗口可以使用 destroy 命令删除 (destroy .path name)。

2-4-12，建立菜单

工具菜单可以使用 Tk 的 menu 命令指定。

```
menu .menu        ;#指定菜单的路径名称
.menu add cascade –label file –menu .menu.file
.menu add cascade –label edit –menu .menu.edit
.menu add cascade –label view –menu .menu.view
                  #指定添加到 .menu 的层叠（文件、编辑、查看）
menu .menu.file –tearoff no
                  #当您在 –tearoff 选项中选取“是”(yes)，所建立的菜单将可
                  #从窗口删除。
.menu.file add command –label exit –command exit
                  #将“exit”分配到 menu.file 作为子菜单
                  #将“exit”定义为选取“file”时的子菜单
. configure –menu .menu
```



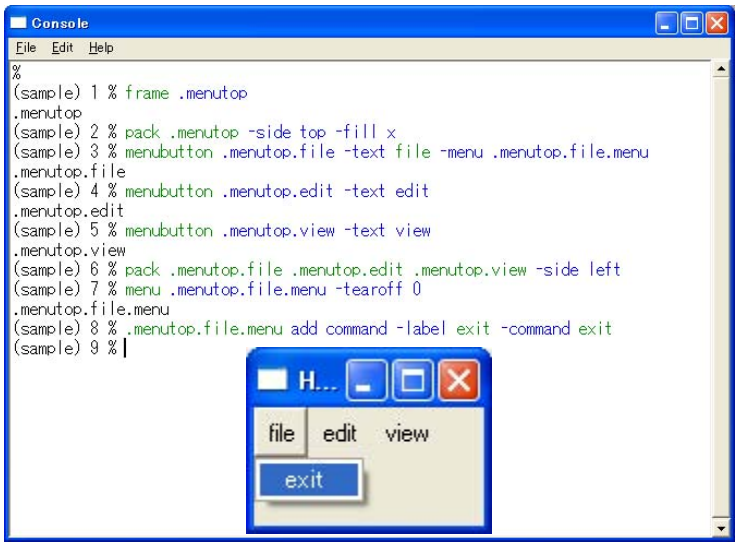
图：工具菜单的建立

下拉式菜单可以使用 menu 命令在上层窗口上建立。

2-4-13，建立菜单按钮

菜单按钮可以使用 Tk 的 menubutton 命令建立。

```
frame .menutop                ;#使用 frame 命令在工具栏上分配框架
pack .menutop –side top –fill x
                              #在 -slide top 中建立的窗口将会分配到上层。
menubutton .menutop.file –text file –menu .menutop.file.menu
                              #在 –menu 选项中将 .menutop.file.menu 分配为子菜单
menubutton .menutop.edit –text edit
menubutton .menutop.view –text view
pack .menutop.file .menutop.edit .menutop.view –side left
                              #分配所建立的菜单按钮
menu .menutop.file.menu –tearoff 0
                              # 在 –tearoff 选项中选取“真”(true) 时，所建立的菜单将可
                              #从窗口删除。
.menutop.file.menu add command –label exit –command exit
                              #定义在选取“menutop.file.menu”时的子菜单。
```



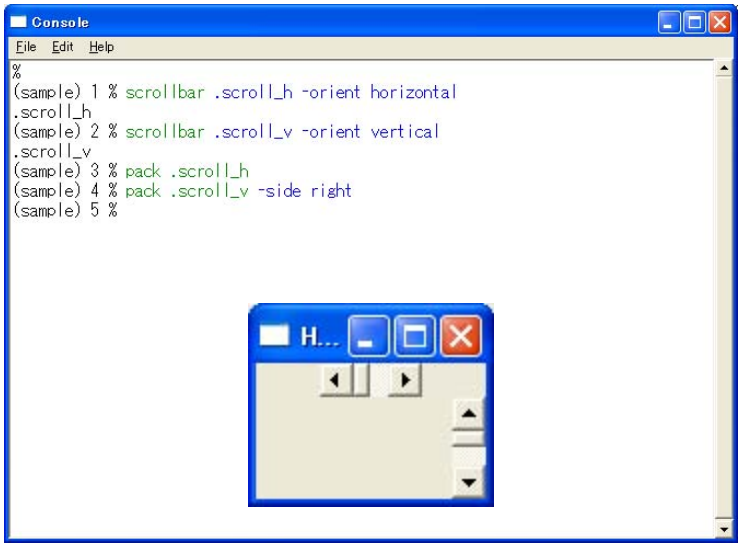
图：菜单按钮的建立

使用 menubutton 命令所建立之菜单的功能与使用 menu 命令建立的一样。如果要在工具栏上分配菜单，必须使用 frame 命令预先分配菜单栏的框架。

2-4-14，建立滚动条

滚动条可以使用 Tk 的 scrollbar 命令在窗口，等等上建立。

```
scrollbar .scroll_h –orient horizontal        ;#定义水平滚动条
scrollbar .scroll_v –orient vertical          ;# 定义垂直滚动条
pack .scroll_h                                ;#分配已定义的滚动条
pack .scroll_v –side right
```



图：滚动条的建立

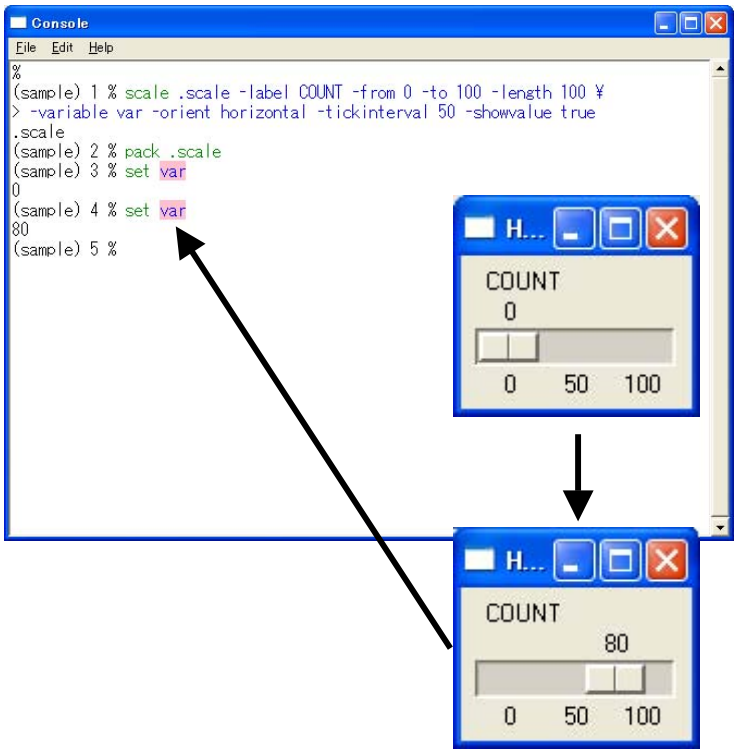
在上述样品程序中，滚动条在上层窗口中分配，也可通过预先建立框架，然后在定义和分配时指定滚动条的路径名称，将滚动条分配到框架，等等上。

```
labelframe .frame –text label –bd 2 –width 100 –height 50 –relief solid
pack .frame

scrollbar .frame.scroll_v –orient vertical     ;#指定框架的路径名称
pack .frame.scroll_v
```

2-4-15，建立用于更改变量值的数值范围栏

用于更改变量的值的数值范围栏可以使用 Tk 的 scale 命令建立。

```
scale .scale –label COUNT –from 0 –to 100 –length 100 \
        －variable var –orient horizontal –tickinterval 50 –showvalue true
pack .scale
```



当指向数值范围栏上的值更改时，它将会反映在 –variable 选项中指定的变量 val 的值中。

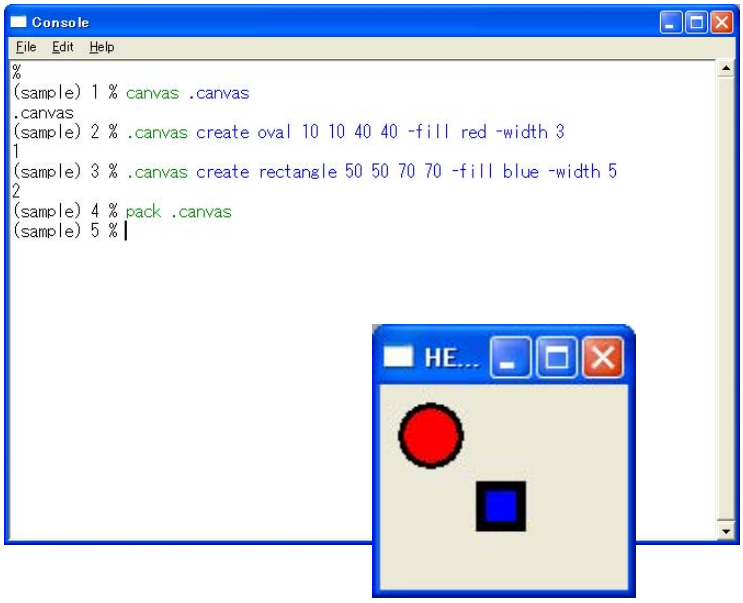
图：数值范围栏的建立

您可以使用 scale 命令的 –from 和 –to 选项指定数值范围栏上的数值范围。数值范围栏上标示的值将会反映在 –variable 选项中指定的变量 val 中。此外，您也可以在 –tickinterval 选项中指定数值范围栏上显示的数值范围指示符的间隔，以及用 –showvalue 选项决定是否在数值范围栏显示数值。

2-4-16，建立画布

具有线条、文本和多边形的画布可以使用 Tk 的 canvas 命令建立。

```
canvas .canvas                          ;#定义您建立的画布
.canvas create oval 10 10 40 40 –fill red –width 3
.canvas create rectangle 50 50 70 70 –fill blue –width 5
pack .canvas                            ;#分配画布
```

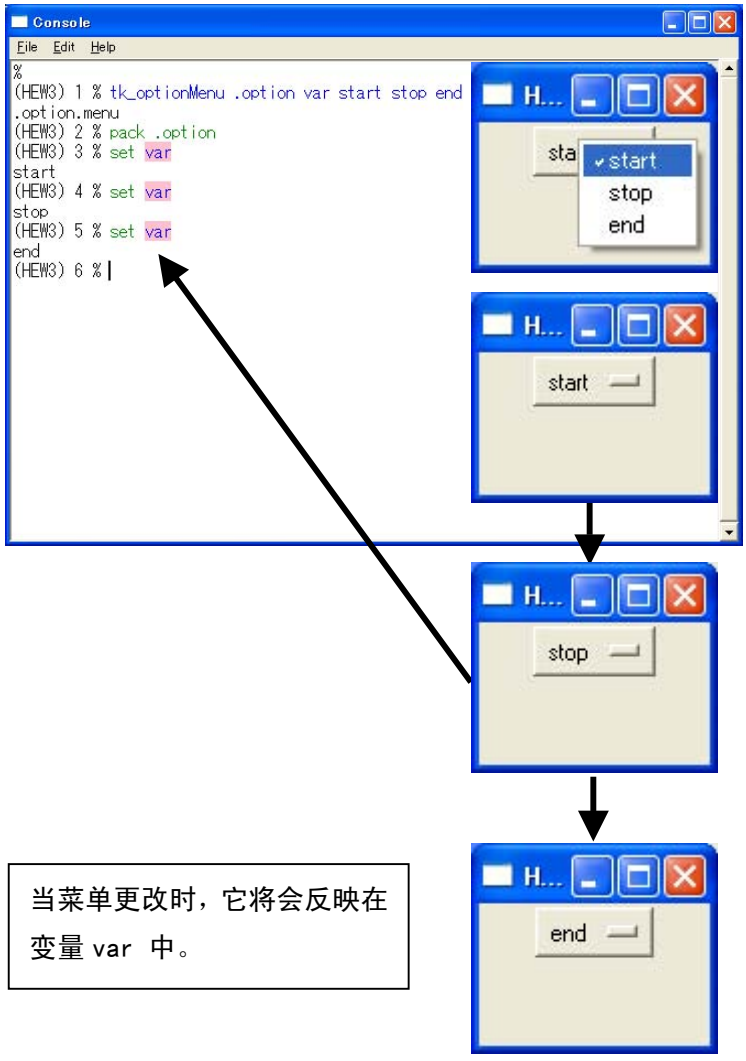


图：画布的建立

2-4-17，建立选项菜单

选项菜单可以使用 Tk 的 tk_optionMenu 命令建立。

```
tk_optionMenu .option var start stop end
pack .option
```



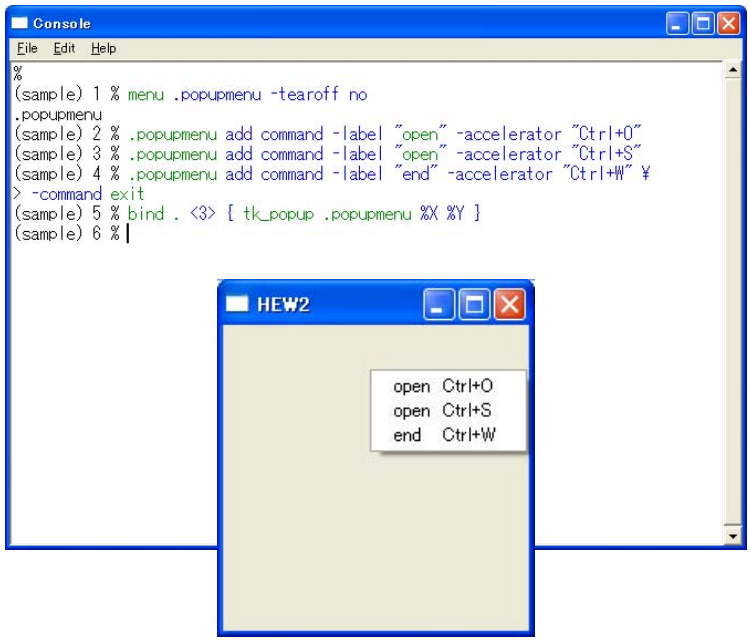
图：选项菜单的建立

在上述样品程序中，变量 var 和选项菜单（“start”、“stop”、“end”）使用 tk_option 命令指定。变量 val 反映三个菜单中其中一个指定的菜单。

2-4-18，建立弹出式菜单

弹出式菜单可以使用 Tk 的 tk_popup 命令建立。

```
menu .popupmenu –tearoff no
        #可通过在 –tearoff 选项中选取“是”(yes) 从窗口删除弹出式菜单。
.popupmenu add command –label “open” –accelerator “Ctrl+O”
.popupmenu add command –label “save” –accelerator “Ctrl+S”
.popupmenu add command –label “end” –accelerator “Ctrl+E” –command exit
        #选取“end”时的子菜单可以在 –command 选项中指定。
bind . <3> { tk_popup .popupmenu %X %Y }
```



图：弹出式菜单的建立

在上述样品程序中，所建立的“.popupmenu”使用 tk_popup 命令定义为弹出式菜单。您可以通过在下列描述中设定 %X 和 %Y 的值，指定弹出式窗口在窗口上的打开位置。

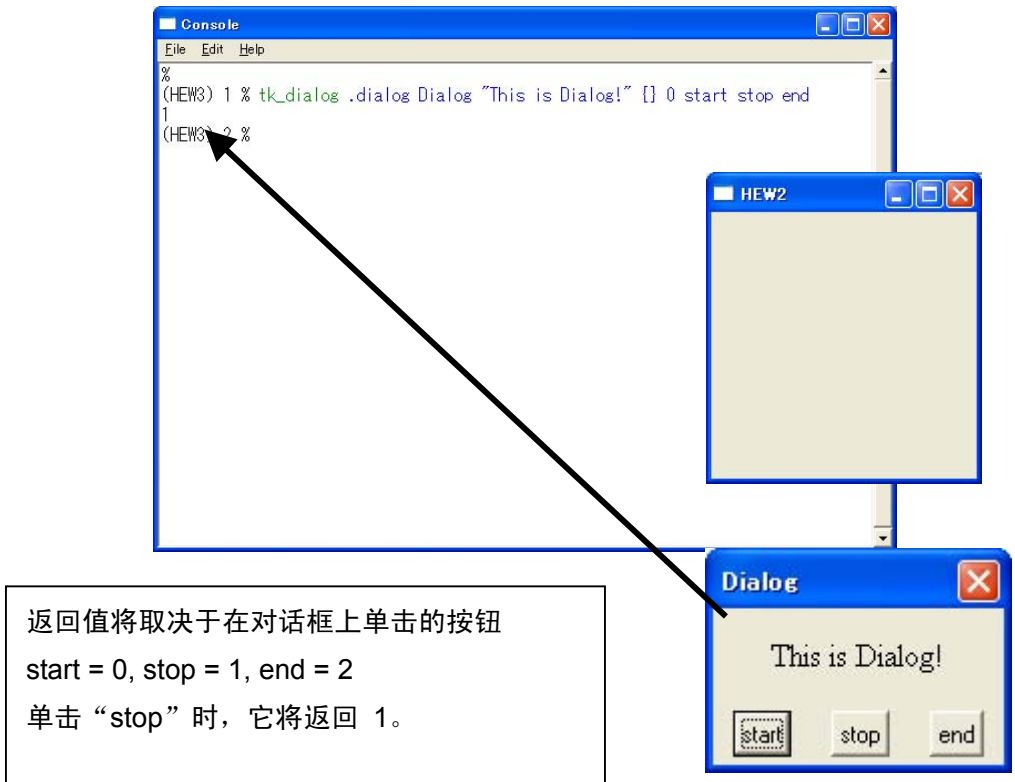
bind .<3> { tk_popup .popupmenu %X %Y }

由此，弹出式菜单可在窗口上的任何位置打开。

2-4-19，建立简单对话框

简单对话框可以使用 Tk 的 tk_dialog 命令建立。

```
tk_dialog .dialog Dialog “This is Dialog!” {} 0 start stop end
```



返回值将取决于在对话框上单击的按钮
start = 0, stop = 1, end = 2
单击“stop”时，它将返回 1。

图：对话框的建立

您可以使用 tk_dialog 命令指定对话框上的窗口标题和信息。在上述样品对话框中，窗口标题命名为“Dialog”，而信息则指定为“This is Dialog!”。

您也可以指定对话框上之按钮的名称及其初始值。在上述样品对话框中，按钮命名为“start”、“stop”和“end”，而其初始值则各指定为“0”、“1”和“2”。

“0”、“1”和“2”的其中一个返回值会在单击其中一个按钮时作为结果返回。

2-4-20，建立信息对话框

信息对话框可以使用 Tk 的 tk_messageBox 命令建立。

```
tk_messageBox –type OK –title message –icon info –message message
```

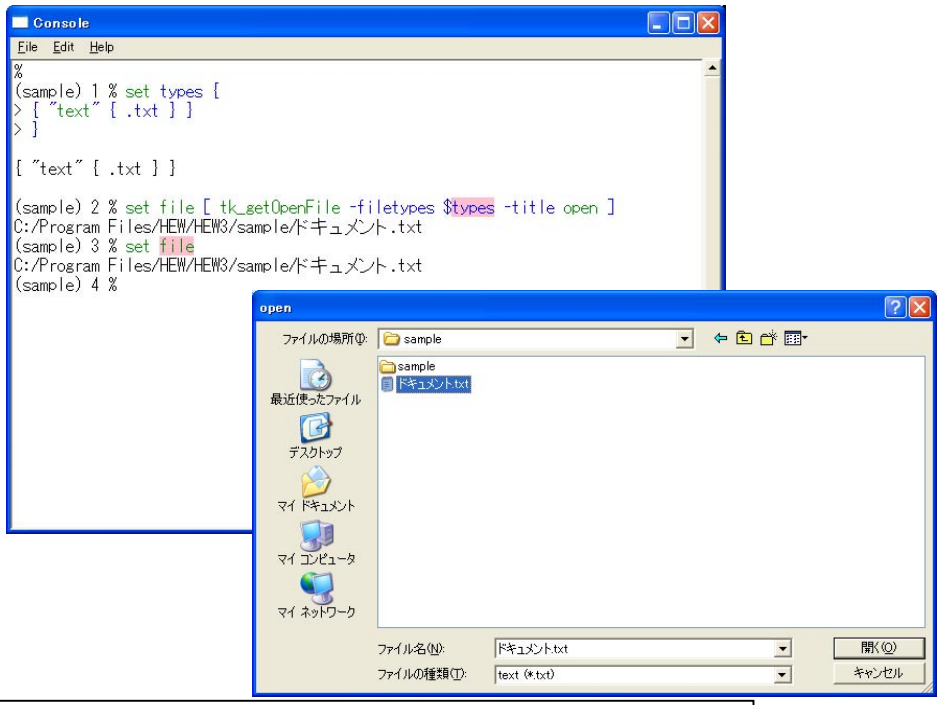
图：信息对话框的建立

您可以使用 tk_messageBox 命令建立交互式信息对话框。您可以在 –type 选项中指定按钮的类型，在 –title 选项中指定按钮的标题，以及在 –message 选项中指定要在对话框上显示的信息。

2-4-21，建立打开现有文件对话框

用于打开现有文件的对话框可以使用 Tk 的 tk_getOpenFile 命令建立。

```
set types {
        { “text” { .txt } }
}
set file [ tk_getOpenFile –filetypes $types –title   open ]
```



选取并打开现有文件时，该文件的存储位置将会反映在 file 变量中。

图：打开文件对话框的建立

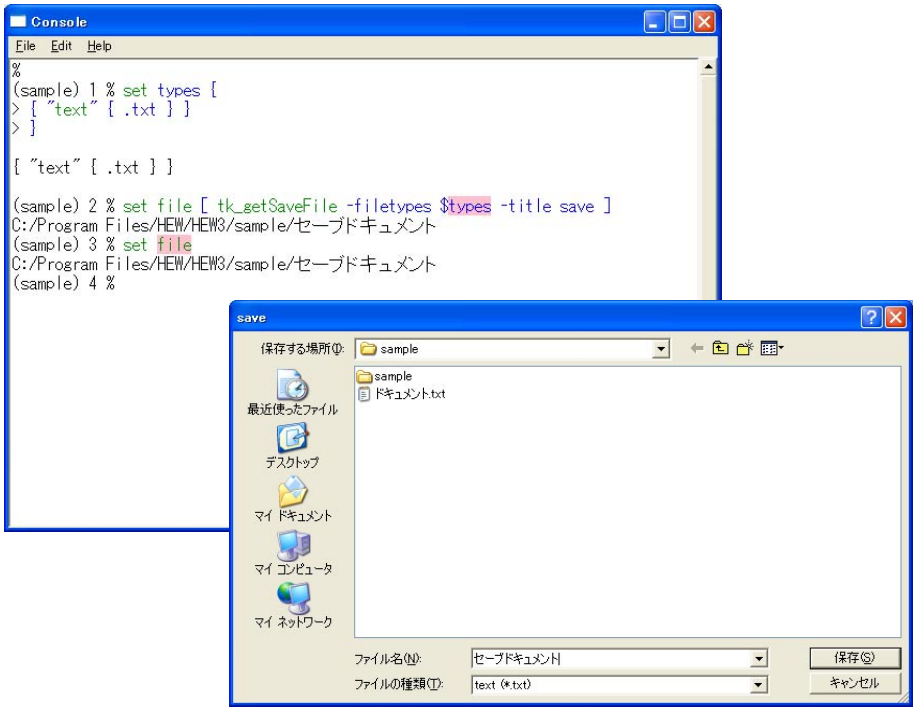
在使用 Tk_getOpenFile 命令建立打开文件对话框时，您可以在 –filetypes 选项中指定文件类型，以及在 –title 选项中指定窗口标题。在上述程序中，类型变量在 –filetypes 选项中选定为文件类型。text [*.txt] 通过预先设定 type{ …} 命令指定为类型变量的文件类型。

此外，选定文件的存储位置将会作为 Tk_getOpenFile 命令的结果返回。
在上述程序中，结果将可以使用 set 命令存储在文件变量中。

2-4-22，建立打开新文件对话框

用于打开和保存新文件的对话框可以使用 Tk 的 tk_getSaveFile 命令建立。

```
set types {
        { “text” { .txt } }
}
set file [ tk_getSaveFile –filetypes $types –title   save ]
```



选取并保存新的文件时，已保存的文件之存储位置将会反映在 file 变量中。

图：保存文件对话框的建立

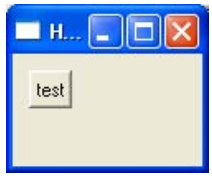
在使用 Tk_getSaveFile 命令建立打开文件对话框时，您可以在 –filetypes 选项中指定文件类型，以及在 –title 选项中指定窗口标题。

此外，选定文件的存储位置将会作为 Tk_getSaveFile 命令的结果返回。

2-4-23，分配窗口小部件

在之前的章节中，仅描述使用 pack 命令来分配窗口小部件的方法。然而，除了 pack 命令，您还可以使用 place 命令和 grid 命令来分配窗口小部件。分配方法因命令而有所不同，因此您可以根据您要建立的 GUI 窗口选择最适当的命令。

分配窗口小部件的实例

命令	使用命令的实例	分配实例
pack	按方向布置窗口小部件。 方向可在 -side [left, right, top, bottom] 选项，等等中设定。	
	pack [button .test –text test –command { go }]	
	pack [button .test –text test –command { go }] –side left	
place	按座标布置窗口小部件。 座标可在 –x 和 –y 选项中指定，而大小则可在 –width 选项中指定。	
	button .test –text test –command { go } place .test –x 10 –y 10	
	button .test –text test –command { go } place .test –x 50 –y 50 –width 100	
grid	将窗口小部件布置为栅格。 栅格位置可在 –column 和 –row 选项中指定，而界限则可在 –padx 和 –pady 选项中指定。	
	button .test –text test –command { go } grid .test	
	button .test –text test –command { go } grid .test –column 3 –row 4 -padx 3 –pady 5	

上表中使用的选项只是众多选项中的一些例子。您可以使用更多选项。请参阅 Tcl/Tk 参考手册等以获得更多信息。

3，在 HEW 中编程 Tcl/Tk

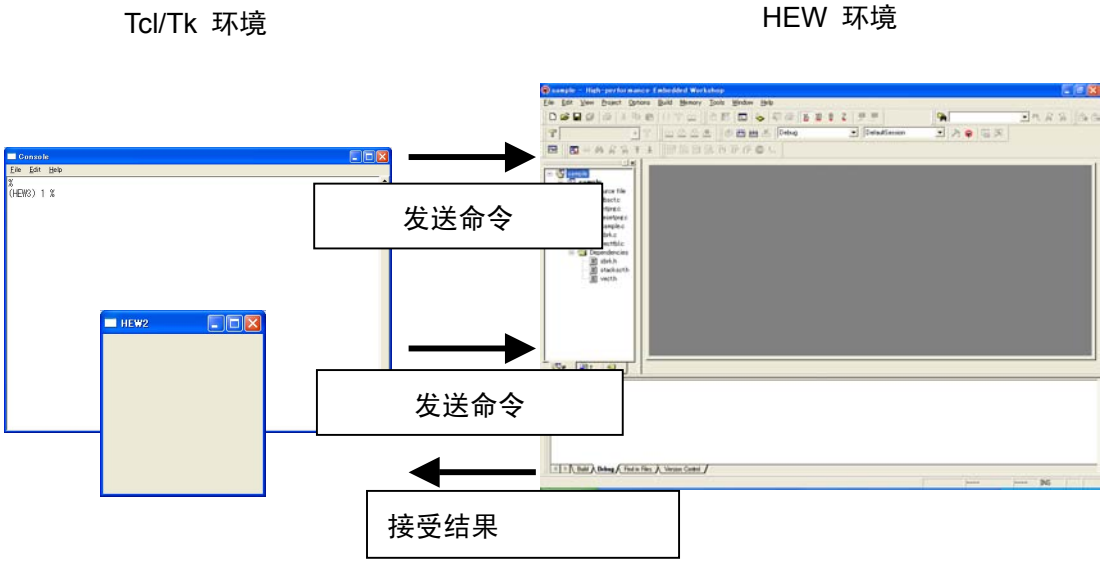
可以在 HEW 中进行 Tcl/Tk 编程。

Tcl/Tk 可通过将 HEW 命令分配到以 Tcl/Tk 建立的 GUI 窗口（如分配到按钮上），建立可满足用户需要的开发环境。

因此，分配到 GUI 窗口上的命令将会发送到 HEW，而 HEW 模拟 (Hew Simulation) 将可受到控制。

下列环境可以通过 Tcl/Tk 建立。

- 从 Tcl/Tk 发送命令到 HEW。
  Tcl/Tk 发送命令到 HEW，以及控制 HEW 的模拟。
- 从 Tcl/Tk 发送命令到 HEW，然后以接受数据作为结果。
  Tcl/Tk 发送命令到 HEW，以及接受结果并显示它们。

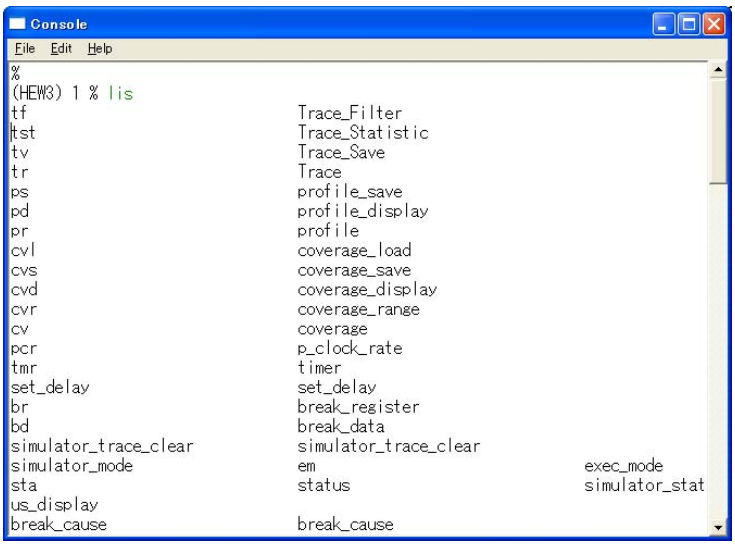


图：HEW 和 Tcl/Tk 之间的连接

3-1，Tcl/Tk 中可用的 HEW 命令

HEW 提供许多命令，可在 HEW 的命令行中使用。您可以使用其中一些命令来建立符合您需要的开发环境。

要参照 Tcl/Tk 中可使用的 HEW 命令列表，可在 Tcl/Tk 的控制台窗口上输入 lis 命令取得。使用前请先查明这些命令的含义。



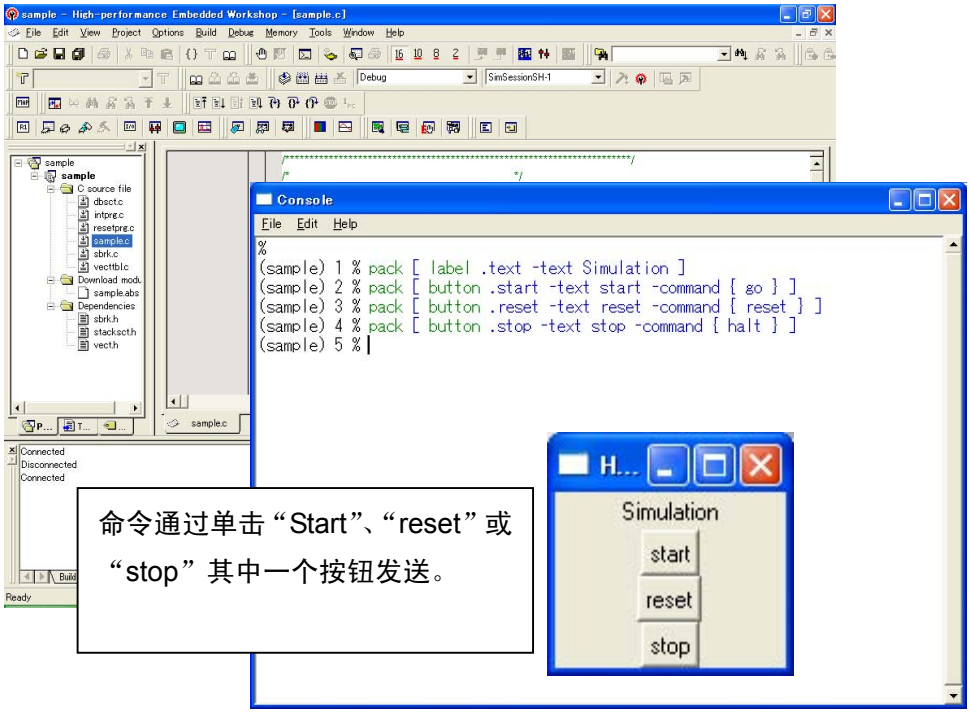
图：控制台窗口上的 HEW 命令列表

3-2，建立控制 HEW 模拟程序的命令

用于控制 HEW 模拟程序的 GUI 窗口可以在 Tcl/Tk 中建立。

```
~ 样品程序 ~
pack [ label .text –text Simulation ]
pack [ button .start –text start –command { go }
pack [ button .reset –text reset –command { reset }
pack [ button .stop –text stop –command { halt }
```

在上述样品程序中，执行 HEW 模拟的最简单方法是在 Tcl/Tk 之 GUI 窗口上分配按钮，控制 HEW 模拟程序的环境便会建立。



图：执行的实例

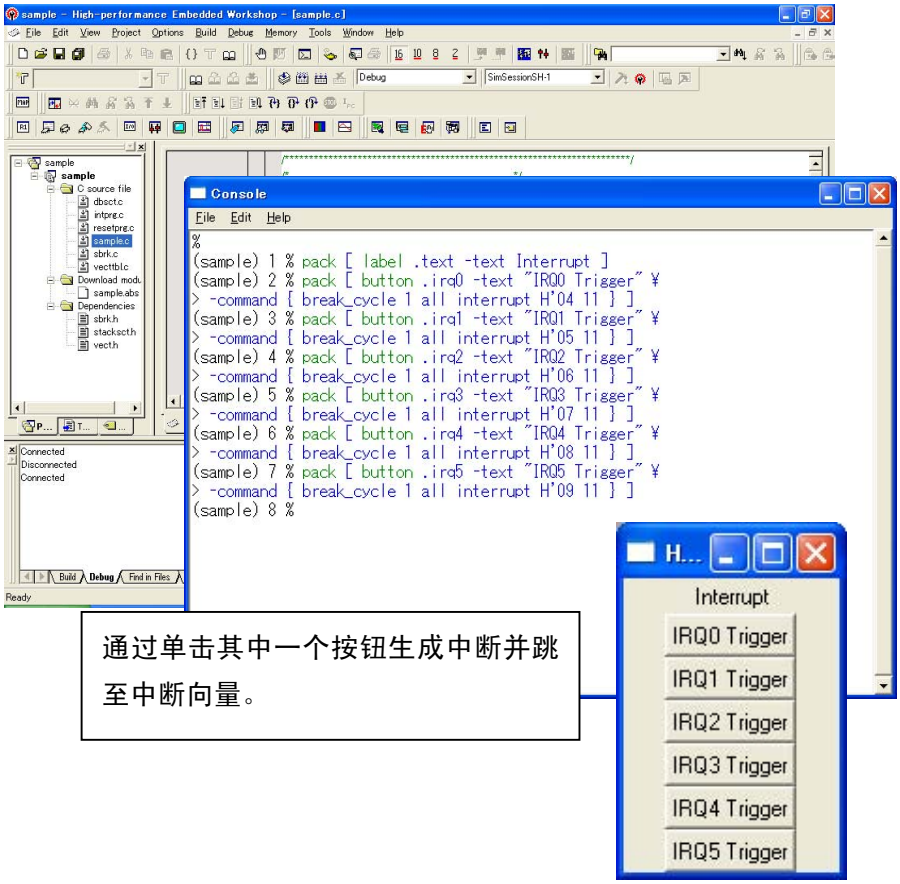
在上述样品程序中，Tcl/Tk 的 GUI 窗口是通过使用 Tcl/Tk 的 label 命令和 button 命令建立。您可以使用 label 命令指定“Simulation”窗口上的信息。您也可以通过指定 –command 选项之花括号中的 HEW 命令，指定要分配到窗口上的按钮，以及要分配到按钮的命令。 因此，取决于在 GUI 窗口上所单击的按钮，将会发送“start”、“reset”或“halt”的其中一个命令。在上述样品程序中，使用 pack 命令分配按钮。

3-3，建立输入类似中断 (Quasi-Interrupt) 到 HEW 模拟程序的命令

用于输入中断信号到 HEW 的 GUI 窗口可以在 Tcl/Tk 中建立。

~ 样品程序 ~

```
pack [ label .text –text Interrupt ]
pack [ button .irq0 –text “IRQ0 Trigger” \
        –command { break_cycle 1 all interrupt H'04 11 }
pack [ button .irq1 –text “IRQ1 Trigger” \
        –command { break_cycle 1 all interrupt H’05 11 }
pack [ button .irq2 –text “IRQ2 Trigger” \
        –command { break_cycle 1 all interrupt H’06 11 }
pack [ button .irq3 –text “IRQ3 Trigger” \
        –command { break_cycle 1 all interrupt H’07 11 }
```



图：执行的实例

在上一页的样品程序中，使用在 HEW 中预备的类似中断函数。通过分配在 GUI 窗口上生成类似中断到按钮的 break_cycle 命令，可以在执行模拟时于任何位置生成中断。
BREAK 命令在 HEW 中预备用于生成类似中断。

BREAK 命令的列表

命令	如何使用命令的实例
break_access	break_access <start_addr> [< end_addr>] [<mode>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_cycle	break_cycle <cycle> [<count>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_data	break_data <addr> <data> [<size>] [<option>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_register	break_register <register> [<data> <size>] [<option>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_point	break_point <addr> [<count>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]

模拟之前，必须将这些命令分配到按钮。

在上一页的样品程序中，break_cycle 1 all …命令已分配到按钮上。（SH1 选定为 CPU）因此，如果在模拟中途单击一个按钮，类似中断将会在单击按钮时的 1 个循环后生成。

3-4，建立控制 HEW 模拟的环境

集成的环境可通过建立脚本文件建立，它由控制 HEW 模拟的命令构建。

```
~ 样品程序 ~
#!/bin/sh
# 下一行使用 wish\ 重新开始
exec tclsh "$0" "$@"

catch {destroy .top}

##########################################################################
####
# 建立窗口小部件        窗口
# HEW 的 false 中断命令描述到各按钮的粗体字母部分。
##########################################################################
####
toplevel      .top
wm title      .top "HEW Simulation"
wm geometry .top 230x450+216+109; update
wm maxsize    .top 1028 753
wm minsize    .top 104 1
##########################################################################
#####
# 设定命令        按钮
# 指定各按钮的排列部分。
##########################################################################
#####
button .top.go      -command {break_clear;go} -height 0 -pady 0 \
          -text {Simulation Go} -width 15
button .top.rego    -command {break_clear;reset;go} -height 0 -pady 0 \
          -text {Reset Simulation} -width 15
button .top.reset -command {reset} -height 0 -pady 0 \
          -text {Reset} -width 15
button .top.irq0    -command {break_cycle 1 all Interrupt H'04 11} -pady 0 \
```

定义新的上层窗口

定义按钮

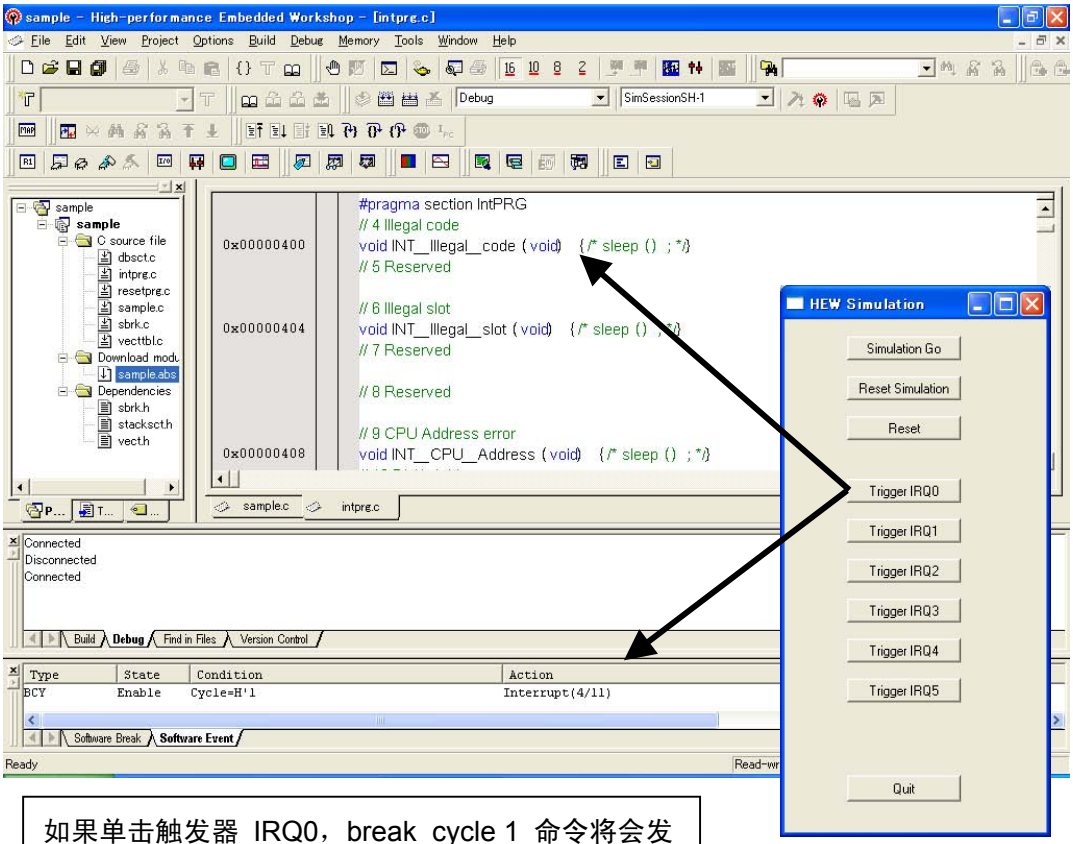
```
button .top.irq2    -command {break_cycle 1 all Interrupt H'06 11} -pady 0 \
          -text {Trigger IRQ2} -width 15
button .top.irq3    -command {break_cycle 1 all Interrupt H'07 11} -pady 0 \
          -text {Trigger IRQ3} -width 15
button .top.irq4    -command {break_cycle 1 all Interrupt H'08 11} -pady 0 \
          -text {Trigger IRQ4} -width 15
button .top.irq5    -command {break_cycle 1 all Interrupt H'09 11} -pady 0 \
          -text {Trigger IRQ5} -width 15
button .top.exit    -command exit \
          -text {Quit} -width 15
#############################################################################
#####                                                    分配按钮
# 设定位置        按钮
#############################################################################
#####
place .top.go       -in .top -x 55 -y 15    -anchor nw -bordermode inside
place .top.rego     -in .top -x 55 -y 50    -anchor nw -bordermode inside
place .top.reset -in .top -x 55 -y 85    -anchor nw -bordermode inside
place .top.irq0     -in .top -x 55 -y 140 -anchor nw -bordermode inside
place .top.irq1     -in .top -x 55 -y 175 -anchor nw -bordermode inside
place .top.irq2     -in .top -x 55 -y 210 -anchor nw -bordermode inside
place .top.irq3     -in .top -x 55 -y 245 -anchor nw -bordermode inside
place .top.irq4     -in .top -x 55 -y 280 -anchor nw -bordermode inside
place .top.irq5     -in .top -x 55 -y 315 -anchor nw -bordermode inside
place .top.exit     -in .top -x 55 -y 400 -anchor nw -bordermode inside
```

如果从 IRQ0 至 IRQ5 的类似中断是使用 HEW 模拟程序中的类似中断函数生成，break 命令的设置将会保留在 HEW 的模拟程序中。若要再次执行模拟，必须关闭 break 命令。在上述样品程序中，除了 break 命令，下列命令也分配到 GUI 窗口上的按钮。

```
button .top.go       -command {break_clear;go} -height 0 -pady 0 \
          -text {Simulation Go} -width 15
button .top.rego    -command {break_clear;reset;go} -height 0 -pady 0 \
          -text {Reset Simulation} -width 15
```

这些命令可通过在 -command 选项的花括号“{}”中列出它们并以分号“;”隔开各命令来进行分配。单击一个按钮时，列出的命令将会从左边开始发送到 HEW。

在上一页的样品程序中，break_clear 命令发送到 HEW 以在执行“go”和“reset go”命令之前关闭 break_cycle 命令。因此，模拟将可以再次执行。



如果单击触发器 IRQ0，break_cycle 1 命令将会发送到 HEW 的 event 点，而类似中断 (Quasi-Interrupt) 将会生成，然后跳至中断向量。

图：执行的实例

3-5，建立向 HEW 模拟输入控制信息的环境 1

用于在执行 HEW 模拟时输入控制命令到 HEW 模拟程序的环境。
控制环境可接受外部输入。

~ 样品程序 ~

```
#!/bin/sh

# 地址的初始设定
set addr ff800030
# 数据的初始设定
set data 0

# 用于地址设置的命令
# 地址值存储在变量 addr 中
label .l_addr –text ADDRESS
place .l_addr –x 10 –y 10

entry .addr –textvariable addr
place .addr –x 70 –y 10

# 用于数据设置的命令
# 数据值存储在变量 data 中
label .l_data –text DATA
place .l_data –x 10 –y 50

entry .data –textvariable data
place .data –x 70 –y 50

# HEW 命令设定为一个按钮
# 按下按钮可将数据设定为任意地址。
button .set -command {memory_fill $addr $addr $data} -text {set data}
place .set –x 60 –y 100
```

设定变量 addr 和 data 的初始值

建立输入地址的输入框

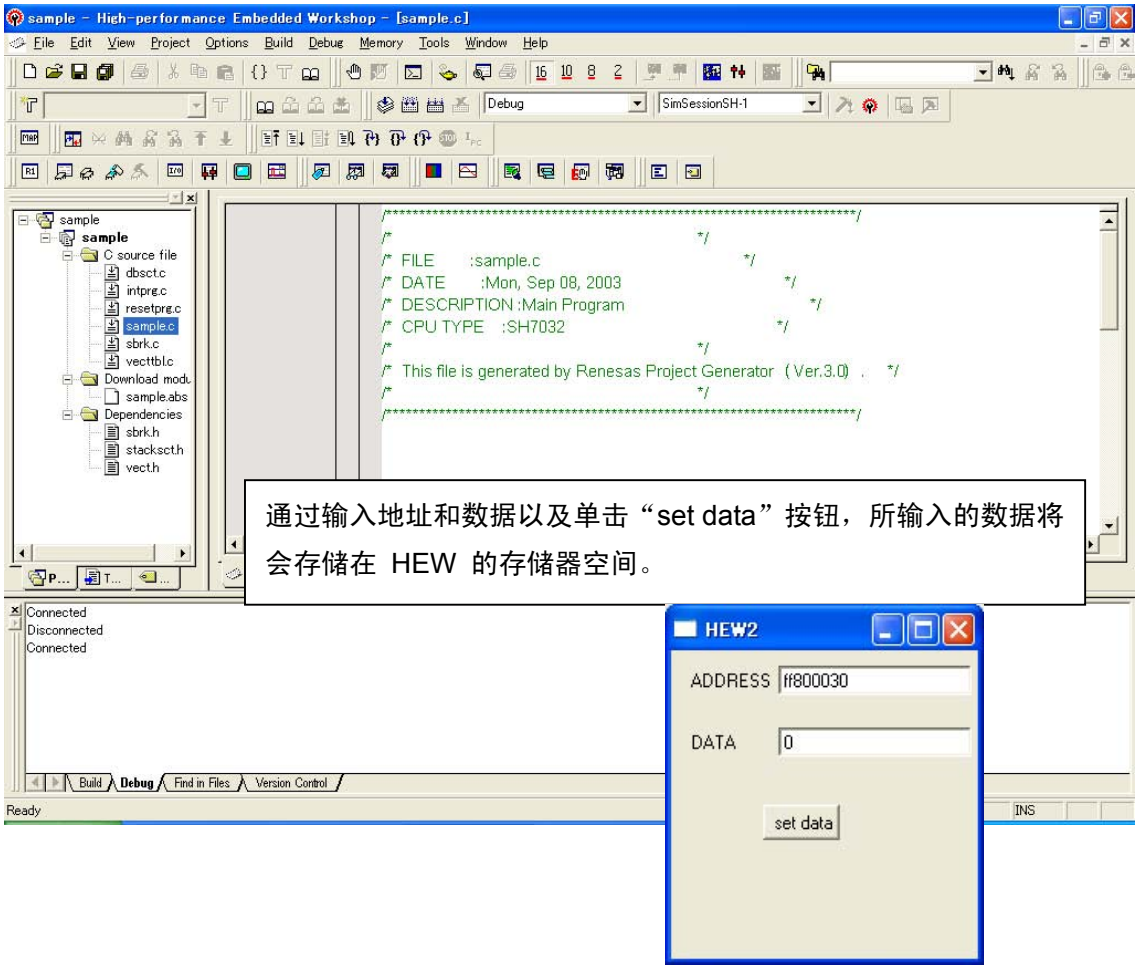
使用 place 命令指定 Label 和 entry 的座标并分配它们。

建立输入数据的输入框

使用 HEW 的 memory_fill 命令指定地址和数据值。
通过 entry 命令指定变量 addr 和 data 的值。

在上一页的样品程序中，建立了一个输入控制命令环境。任何地址和数据都可以使用输入框在 HEW 中存储。当地址和数据以十六进制在 GUI 窗口上输入以及单击“set data”按钮时，数据将会存储在 HEW 存储器空间中的指定地址。

数据可以使用 memory_fill 命令存储。发送到 HEW 的 memory_fill 命令可在 button 命令的 -command 选项中选取。Memory_fill 命令通过参考 entry 命令中指定的变量 addr 和数据（“$”会在参考数据时使用），将地址和数据值传递到 HEW。



图：执行的实例

3-6，建立向 HEW 模拟输入控制信息的环境 2

用于在 HEW 的存储器空间或寄存器中设定值的环境可以在 Tcl/Tk 中建立。

```
~ 样品程序 ~
wm geometry . 350x280                          现有窗口的大小改变

# 数据
scale .scale -label DATA -from 0 -to 10000 -length 300 -variable var -orient horizontal
-tickinterval 2500 -showvalue true
place .scale -x 0 -y 0                         定义输入数据的数值范围栏

# 地址
label .l_addr -text ADDRESS
place .l_addr -x 10 -y 100                     定义输入地址的输入框

entry .addr -textvariable addr
place .addr -x 70 -y 100

# 数据大小
label .l_size -text SIZE
place .l_size -x 10 -y 140                     定义输入数据的单选按钮

set select size8
set select_size "BYTE"
radiobutton .size8     -text 8    -variable select -value size8
radiobutton .size16 -text 16 -variable select -value size16
radiobutton .size32 -text 32 -variable select -value size32

place .size8 -x 60 -y 140
place .size16 -x 100 -y 140
place .size32 -x 140 -y 140
```

```
set addr     0
set addr_s 3
set addr_e 3
set var16    0
set var16_tmp 0

# 数据修改  proc
proc set_data { } {
        global var
        global var16
        global var16_tmp
        global select
        set var16_tmp [ format %08x $var ]

        if { $select == "size8" } {
                set var16 [ string range $var16_tmp 6 7 ]
        } elseif { $select == "size16" } {
                set var16 [ string range $var16_tmp 4 7 ]
        } else {
                set var16 $var16_tmp
        }
}

# 数据大小  proc
proc set_size { } {
        global select_size
        global select
        if { $select == "size8" } {
                set select_size "BYTE"
        } elseif { $select == "size16" } {
                set select_size "WORD"
        } else {
                set select_size "LONG"
        }
}
```

指定每个变量的初始值

定义处理通过指定数据大小存储在 HEW 之数据的 procedure。

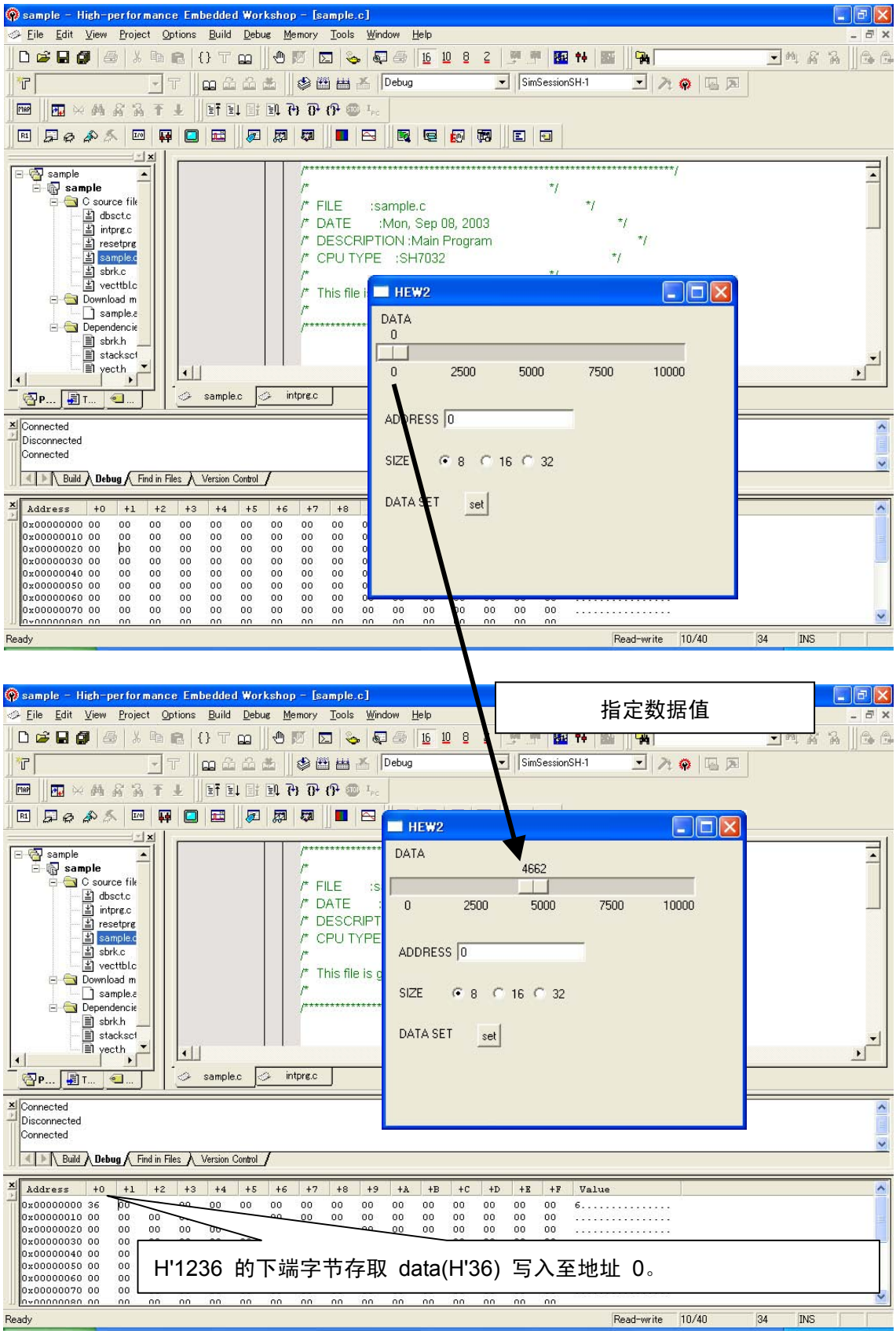
定义生成通过指定数据大小发送到 HEW 之命令的参数（数据大小）的 procedure。

```
# 起始地址 proc
proc set_addr_s { } {
        global select
        global addr
        global addr_s
        if { $select == "size8" } {
                set addr_s [ expr $addr + 3 ]
        } elseif { $select == "size16" } {
                set addr_s [ expr $addr + 2 ]
        } else {
                set addr_s [ expr $addr + 0 ]
        }
}

# 终止地址 proc
proc set_addr_e { } {
        global select
        global addr
        global addr_e
        if { $select == "size8" } {
                set addr_e [ expr $addr + 3 ]
        } elseif { $select == "size16" } {
                set addr_e [ expr $addr + 3 ]
        } else {
                set addr_e [ expr $addr + 3 ]
        }
}

# 按钮
label .l_data -text "DATA SET"
place .l_data -x 10 -y 180

button .set -text set -command { set_
$addr_s $addr_e $var16 $select_size }
place .set -x 90 -y 180
```

定义生成通过指定数据大小发送到 HEW 之命令的参数（起始地址）的 procedure。

定义生成通过指定数据大小发送到 HEW 之命令的参数（终止地址）的 procedure。

定义“set”按钮
procedure call 命令和 memory_fill 命令分配到按钮。


sample - High-performance Embedded Workshop - [sample.c]
HEW2
DATA
0
2500
5000
7500
10000
ADDRESS
SIZE
8
16
32
DATA SET
set
指定数据值
4662
H'1236 的下端字节存取 data(H'36) 写入至地址 0。

图：执行的实例

该值必须在样品程序中以十进制（模拟值）指定。在输入地址值和指定数据大小之后单击“set”按钮时，数据将会存储在指定的地址。

如果选取 SIZE 8 和写入 4662(H'1236) 则下端字节 H'36 将设定为地址 0。如果选取 SIZE 16，字 H'1236 将会设定为地址 0。如果选取 SIZE 32，则长字 H'00001236 将会设定为地址 0。

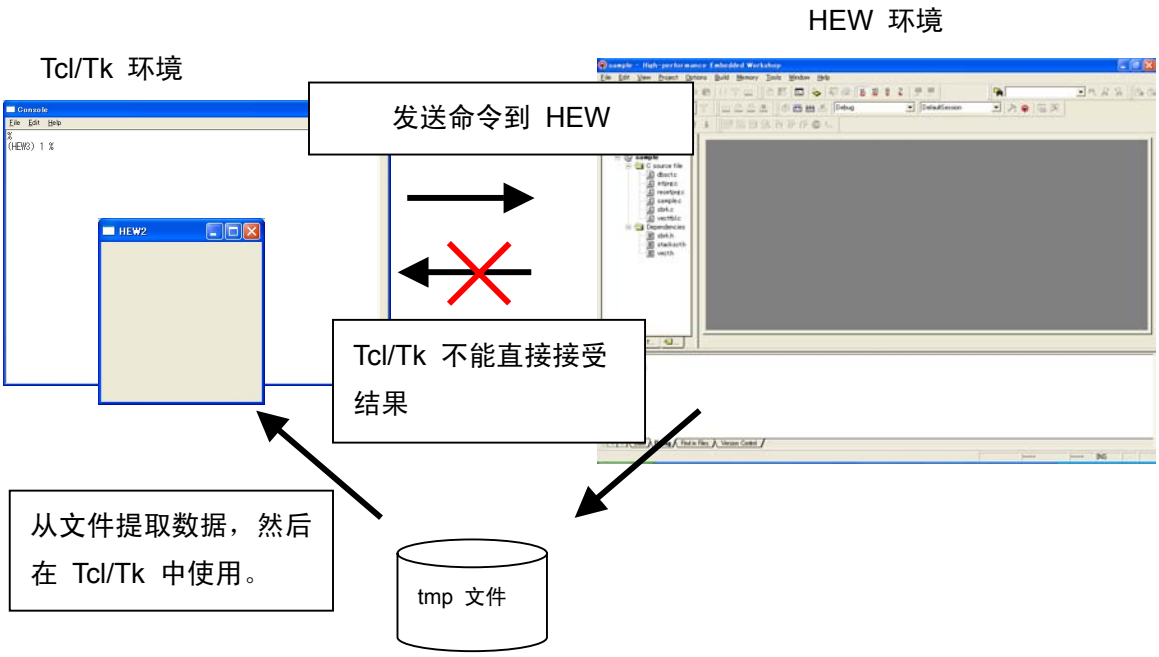
样品程序使用数据、地址和数据大小，以及将这些值设定到 HEW 的“set”按钮的设定进行配置。

此外，处理数据参数、起始地址和终止地址的 procedure 可使用 proc 命令定义。通过使用 button 命令之 –command 选项中的这些 procedure，将数据设定到 HEW 所需的参数将会在单击“set”按钮时生成。

所有 procedure 都简单配置，而且它们不会处理参数的传递。
Procedure 可以通过全局定义 Tcl/Tk 编程中使用的变量进行参考。

3-7，建立从 HEW 模拟输出控制信息的环境 1

用于输入控制命令到 HEW 模拟的环境可通过将具有命令的简单脚本文件建立到 HEW 进行建立。这是因为只有从 Tcl/Tk 到 HEW 的单向命令发送。然而，在从 HEW 模拟输出控制信息的环境下，Tcl/Tk 不可以向 HEW 指派命令而直接接收运行结果。该结果必须通过变量传递到 Tcl/Tk。



图：从 HEW 模拟程序输出控制信息

用于暂时存储数据的文件“tmp.file”（已发送命令的结果）必须预先建立。Tcl/Tk 可通过参考文件中存储的信息接受数据。

3-8，建立从 HEW 模拟输出控制信息的环境 2

本章节将描述从 HEW 建立输出控制信息之环境的步骤，它可直接从 HEW 模拟程序接受在执行模拟时已发送命令的结果。在此环境下，Tcl/Tk 可以接受存储器和寄存器的值并输出它们。

```
~ 样品程序 ~
set content "Display Address "

destroy .b
button .b -text "Update" -command   {

        # 用于在 TEMP 目录中检查“log.txt”
        global env
        set dir $env(TEMP)
        set dataFile [open $dir/log.txt {RDWR TRUNC CREAT}]
        close $dataFile

        # 使用 HEW CommandLine 命令读取地址 0x00000000 的命令
        # 行命令
        md 0 1
        # 已描述要参考此处之地址的 md 命令。

        # 读取为值的控制台文本将记录到 env. variable TMP 指向之
        # 目录的“log.txt”中
        set dataFile [open $::env(TEMP)/log.txt RDWR]
        set b ""
        seek $dataFile 0 start
        set formatted [format "%08X" 0]
        while {$b!= $formatted}   {
                set e [gets $dataFile]
                set d [split $e ""]
                set b ""
                for {set j 0} {$j< 8} {incr j} {append b [lindex $d $j]}
        }
        set ar [split $e]
        set content [lindex $ar 2]
        close $dataFile
```

打开文件选取

从 tmp 文件检索数据

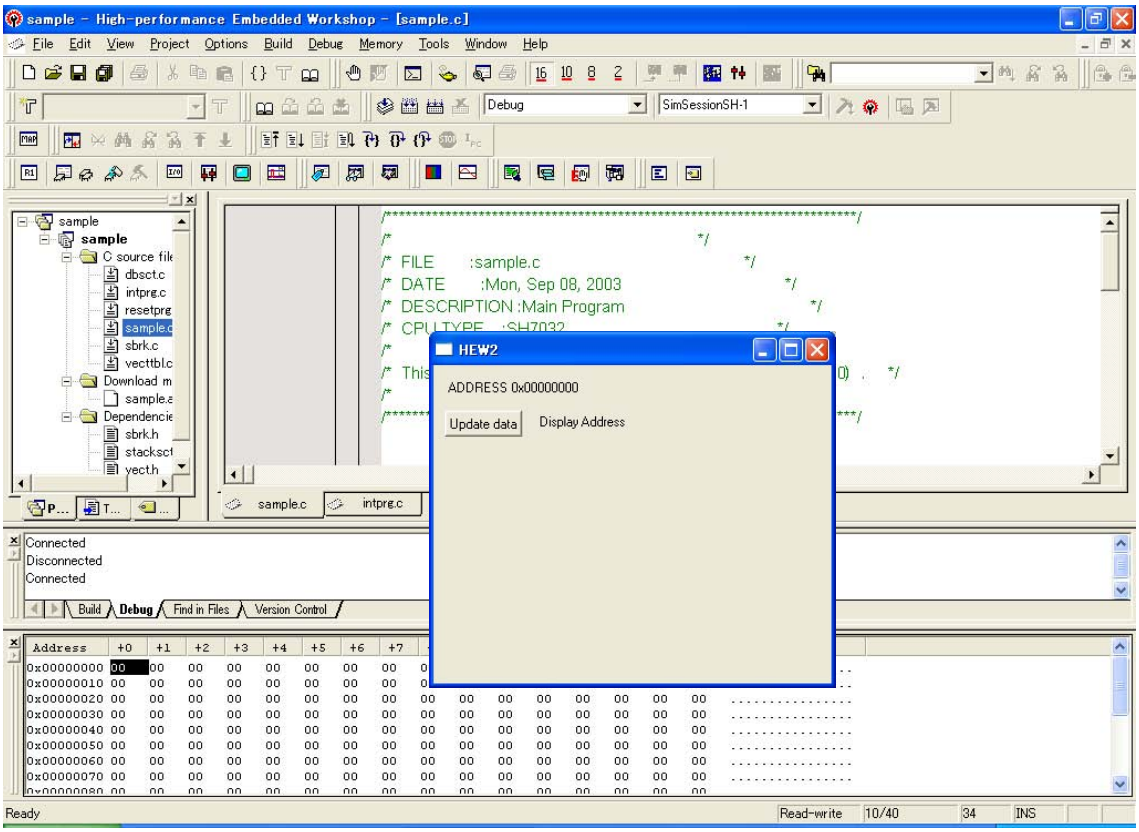
Append 命令将字符串 [lindex $d $j]（取得 d 的 jth 元素的 lindex 命令）添加到字符串 b。

```
                puts -nonewline "Content in Address 0x00000000: 0x"
                puts $content

                destroy .labelcontent
                label .labelcontent -text $content
                place .labelcontent -x 90 -y 10
        }
```

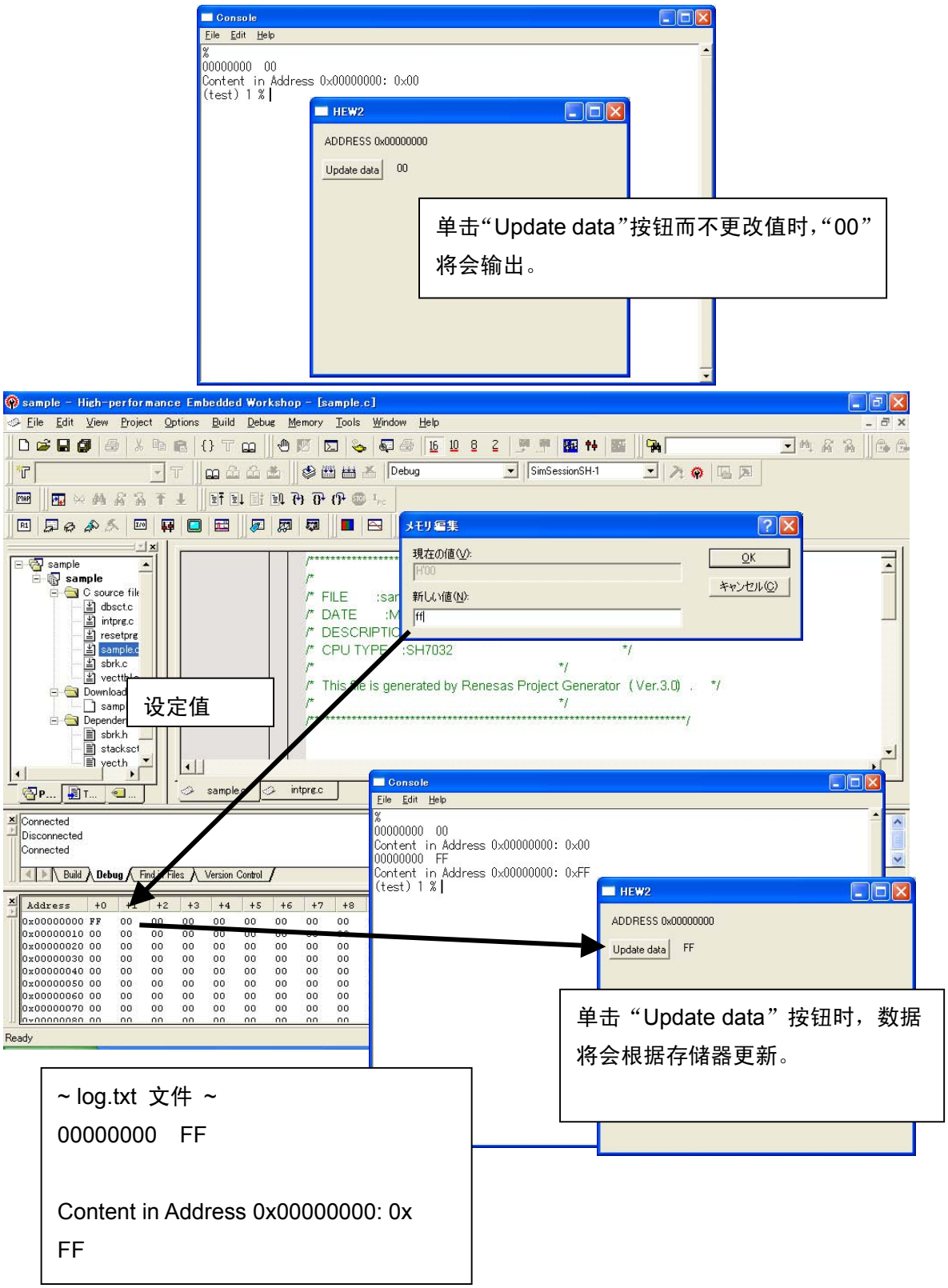
Destroy 命令可作废之前的结果，以及输出更新的数据。

```
destroy .labelcontent
label .labelcontent -text $content

place .labelcontent -x 90 -y 10

place .b -x 10 -y 10
```

分配按钮



图：启动显示


Console
File Edit Help
%
00000000 00
Content in Address 0x00000000: 0x00
(test) 1 %
HEW2
ADDRESS 0x00000000
Update data
00
单击“Update data”按钮而不更改值时，“00”将会输出。
sample - High-performance Embedded Workshop - [sample.c]
File Edit View Project Options Build Debug Memory Tools Window Help
メモリ編集
現在の値(V):
新しい値(N):
OK
キャンセル(C)
设定值
Console
00000000 00
Content in Address 0x00000000: 0x00
00000000 FF
Content in Address 0x00000000: 0xFF
(test) 1 %
HEW2
ADDRESS 0x00000000
Update data
FF
单击“Update data”按钮时，数据将会根据存储器更新。
~ log.txt 文件 ~
00000000 FF
Content in Address 0x00000000: 0x FF

图：执行的实例