To our customers,

# Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to 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.

Application Note

# HEW Tcl/Tk

## Application Note

Renesas Electronics
www.renesas-electoronics.com

Rev.1.00 2003.11

## Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com )
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

# HEW Tcl/Tk
# Application Note

Rev.1.0

# Content

# Tcl/Tk Overview in HEW

A script language, Tcl/Tk is supported in HEW (High-performance Embedded Workshop) The targeted version is Tcl/Tk version 8.4.1.

Tcl/Tk is comprised of class member “Tcl” (Tool Command Language) of script language and “Tk” (Tool Kit), which is used to program graphical user interface.  The script language Tcl/Tk do not need compiling, and the results of execution of the program are reflected immediately.

Tcl has a grammar, which makes simple programming possible. Tcl is used for an application of Stand Alone, and can be built in application programs.
Tk makes it possible constructing GUI suits the needs of users promptly.

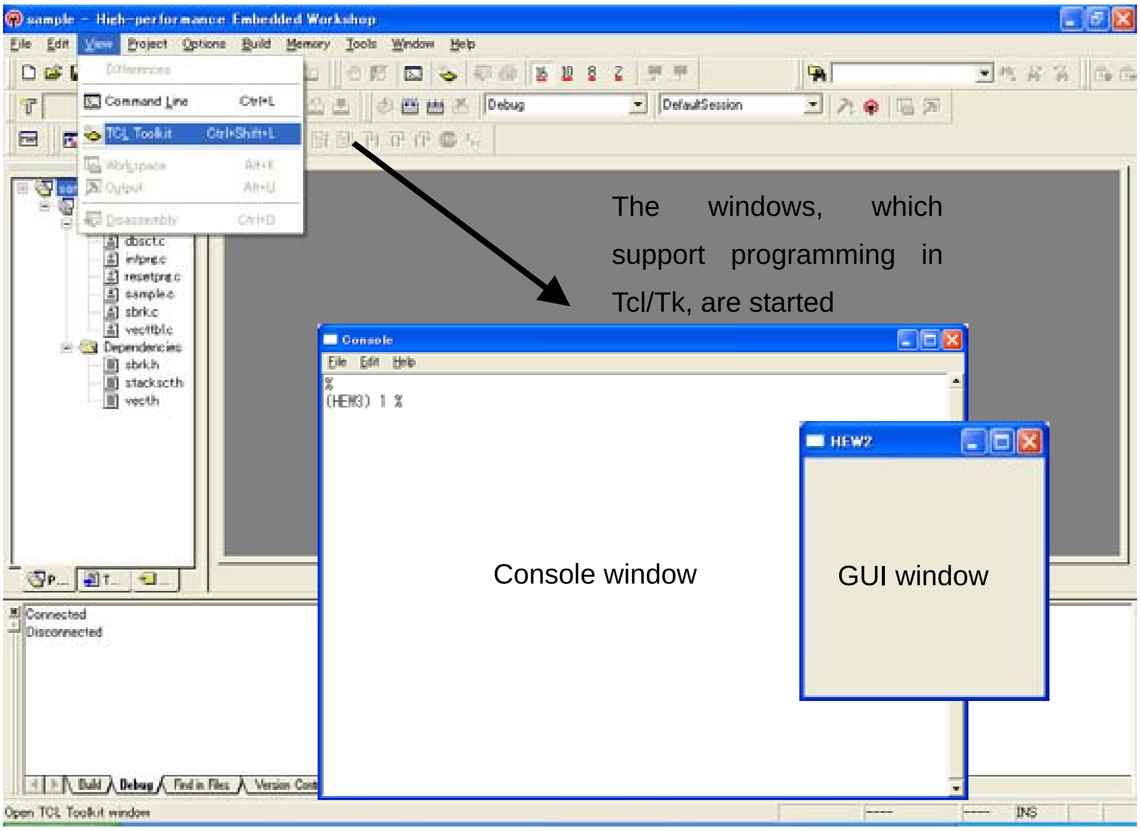
HEW supports Tcl/Tk.  The functions and GUI prepared by default in HEW are usable, and also GUI environment can be customized to meet the users’ individual needs by programming in Tcl/Tk.

# 1, Use of Tcl/Tk

## 1-1,Start up of Tcl/Tk

This chapter describes how to use Tcl/Tk commands in HEW.

Select “View” – “TCL Tool Kit” from HEW command menu.  Console window and GUI window, which supports programming in Tcl/Tk are started.  You can program in Tcl/Tk using these windows.



```
%
(HEW3) 1 %
```

The windows, which support programming in Tcl/Tk, are started

Console window

GUI window

Figure: Start up of Tcl/Tk

Console window

You can program in Tcl/Tk and execute it using interpreter on Console window.  You can also load script files made in advance and execute them.

GUI window

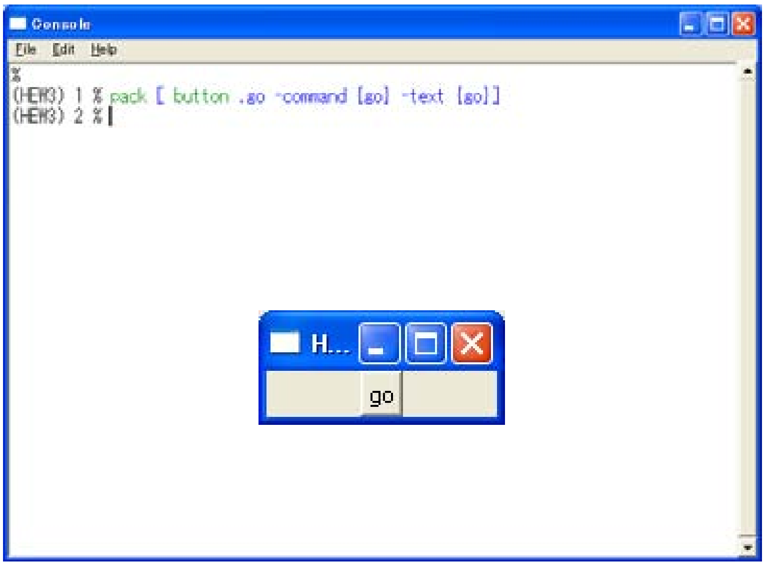
The result of execution of the program on Console window is reflected on GUI window. Toplevel window is prepared in HEW by default.  Additionally, you can create GUI window newly aside from Toplevel window.

## 1-2, Execution method of Tcl/Tk

You can execute Tcl/Tk on HEW according to the following 3 methods.

① Interpreter method

You can assign commands of Tcl/Tk on Console window, when you program in interpreter method. The assigned commands of Tk are reflected on GUI window at the time of Tcl/Tk Tool Kit is started up.



```
%
(HEW3) 1 % pack [ button .go -command [go] -text [go]]
(HEW3) 2 %
```

For example: In case of that a program execution command of HEW “go” is assigned on a button.

Figure: Programming in interpreter method

② Execution of loaded script file

You can program in Tcl/Tk using script file in advance, and load the file on console window. “Select a file to source” window is opened by selecting “File” – “Source” from command menu of Console window. Specify the Tcl/Tk script file created in advance after the window is opened.

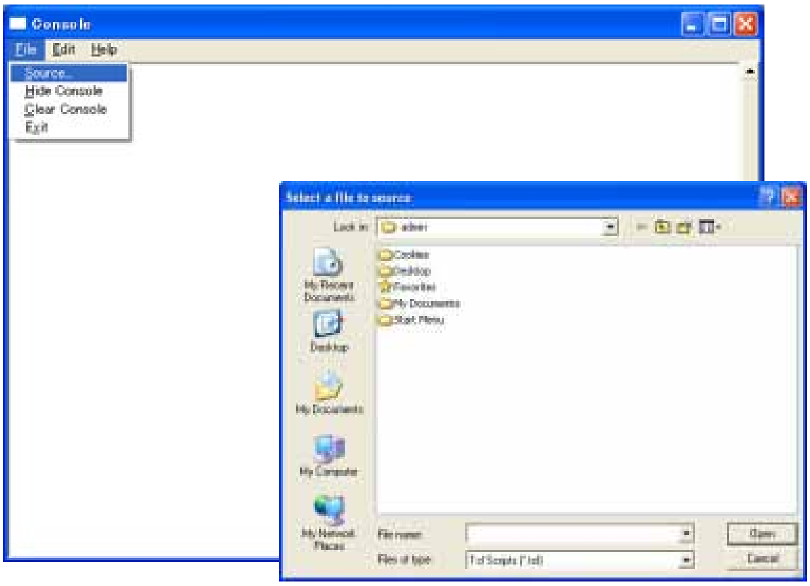
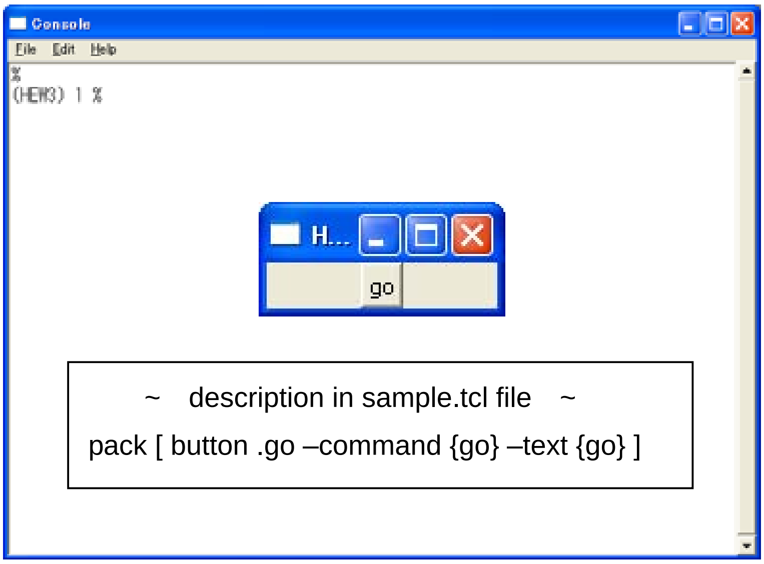


Figure: Tcl/Tk source file selection



```
%
(HEW3) 1 %
```

~ description in sample.tcl file ~

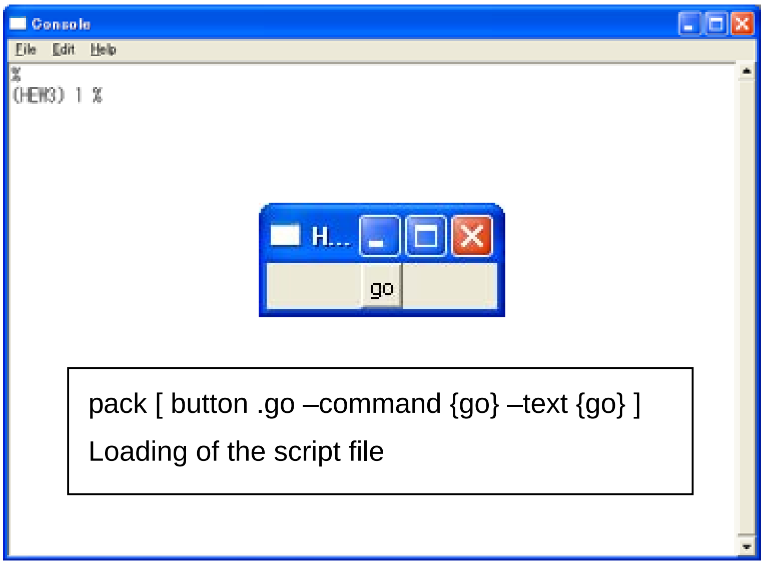
pack [ button .go –command {go} –text {go} ]

Figure: After source file selection

Programming in the script file is reflected on Console window after source file selection.

③ Execution using both of interpreter method and script load method

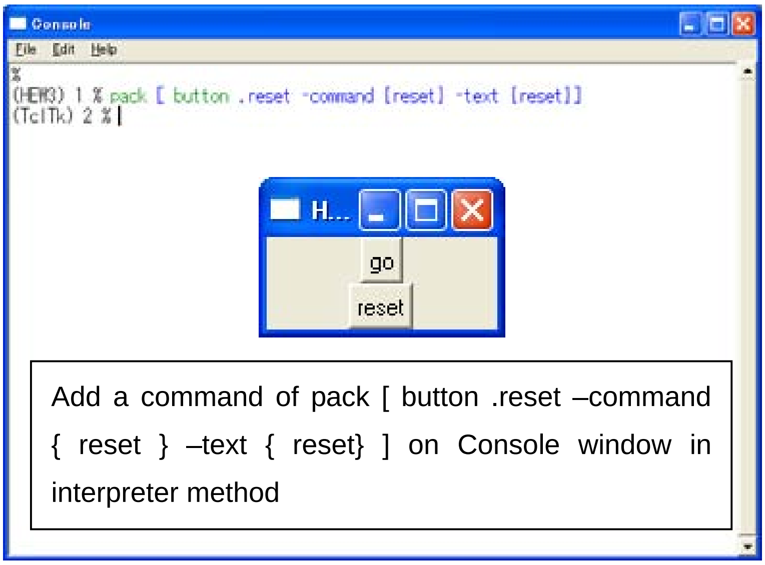
After loading a script file created in advance, you can add programs to the script file in interpreter method.



```
%
(HEW3) 1 %
```

pack [ button .go –command {go} –text {go} ]

Loading of the script file



```
%
(HEW3) 1 % pack [ button .reset -command [reset] -text [reset]]
(TclTk) 2 %
```

Add a command of pack [ button .reset –command { reset } –text { reset} ] on Console window in interpreter method

For example: Add reset command of HEW “reset”

Figure: Program addition

# 2, Basic Programming method in Tcl/Tk

## 2-1, Programming method in Tcl

### 2-1-1,Basic Grammar

Tcl has very simple grammar.  Built-in commands of Tcl and procedures created by user are used as commands in Tcl programming.  You can separate a command and arguments required for the command with a space, and construct one execution format.  You can also separate them by starting a new line, or with a semicolon in order to describe complicated execution format.

Basic Grammar 1：List arguments by separating with spaces.

```
Command arg1 arg2 arg3 …
```

Basic Grammar 2：Separate with semicolon(;) in case of listing several commands on one line.

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

Basic Grammar３：Use ¥ sign in case of listing one command in a few lines.

```
Command arg1 ¥
        arg2 arg3…
```

Basic Grammar 4：  Use # sign in case of describing comments.

```
Command arg1                    #comments
```

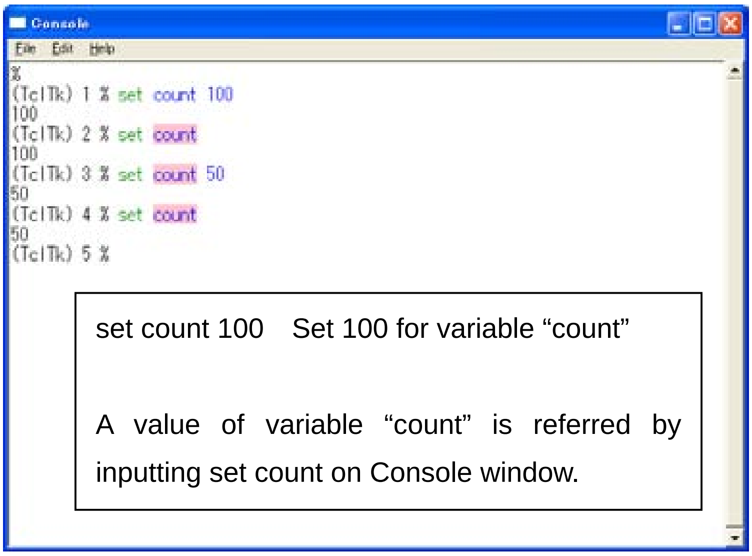
Build-in command of Tcl and procedure created by user are used as commands.
Arg is argument, which is required for built-in command of Tcl and procedures made by user.

## 2-2, Built-in command of Tcl

This chapter describes built-in commands prepared in Tcl. Built-in commands introduced in this chapter are minimum necessary for programming.

### 2-2-1, Variable

Variables in Tcl do not need declaration of the types before using. You can name them as you want. You can also set a value to the variable, and refer to a value of the variable in Tcl programming.



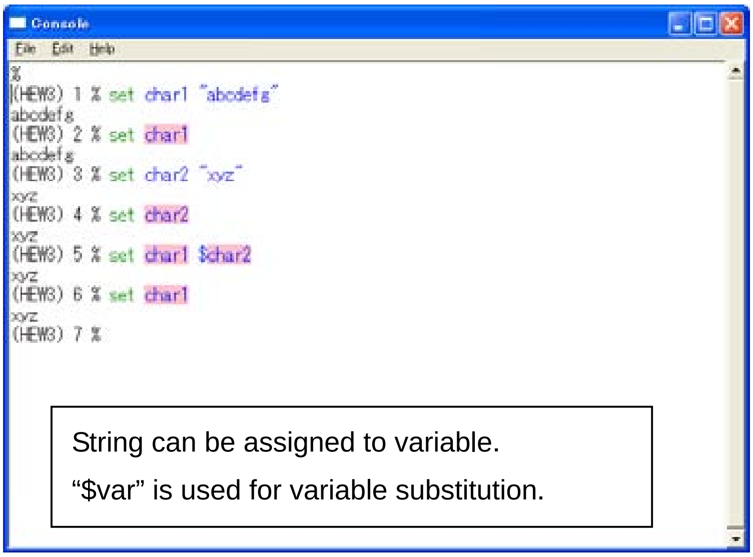
```
%
(TclTk) 1 % set count 100
100
(TclTk) 2 % set count
100
(TclTk) 3 % set count 50
50
(TclTk) 4 % set count
50
(TclTk) 5 %
```

set count 100 Set 100 for variable “count”

A value of variable “count” is referred by inputting set count on Console window.

Figure: Setting and referring of variable

You can also set string for variable as you want in Tcl



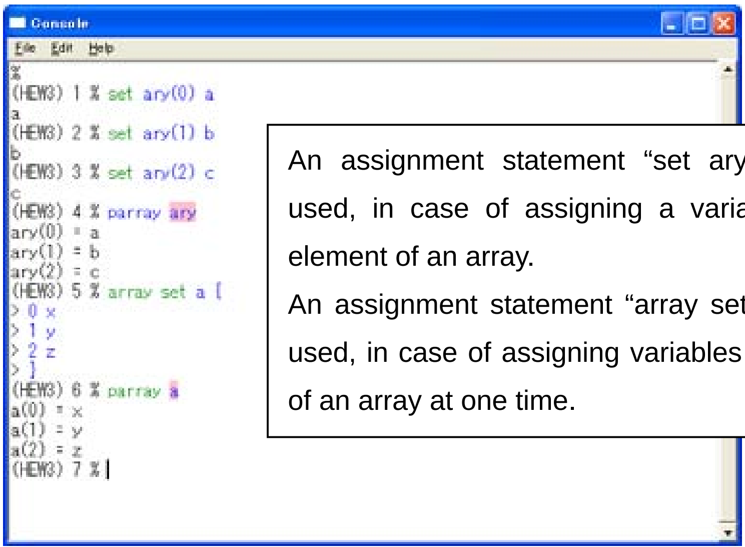
```
%
(HEW3) 1 % set char1 "abcdefg"
abcdefg
(HEW3) 2 % set char1
abcdefg
(HEW3) 3 % set char2 "xyz"
xyz
(HEW3) 4 % set char2
xyz
(HEW3) 5 % set char1 $char2
xyz
(HEW3) 6 % set char1
xyz
(HEW3) 7 %
```

String can be assigned to variable.
“$var” is used for variable substitution.

Figure: Assignment of string to variable

### 2-2-2, Array

Array can be used in Tcl.　You can assign variable to each element of an array, and also to the entire of an array at one time.



```
%
(HEW3) 1 % set ary(0) a
a
(HEW3) 2 % set ary(1) b
b
(HEW3) 3 % set ary(2) c
c
(HEW3) 4 % parray ary
ary(0) = a
ary(1) = b
ary(2) = c
(HEW3) 5 % array set a [
> 0 x
> 1 y
> 2 z
> ]
(HEW3) 6 % parray a
a(0) = x
a(1) = y
a(2) = z
(HEW3) 7 %
```

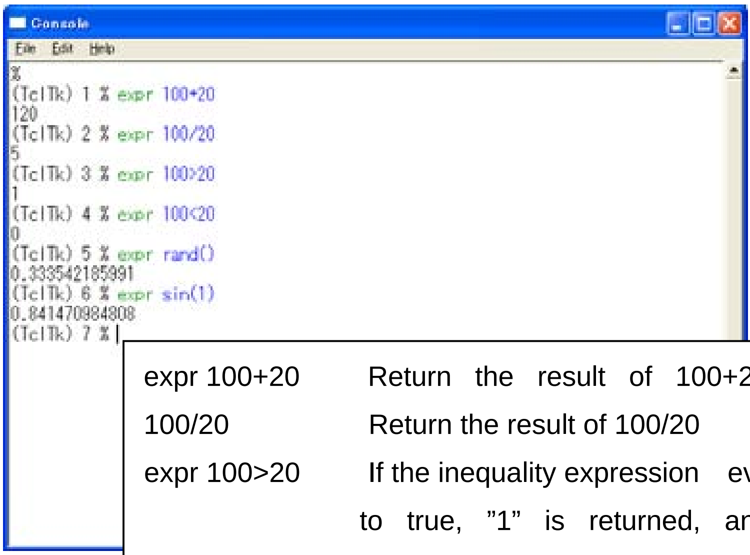
An assignment statement “set ary(0)” can be used, in case of assigning a variable to each element of an array.

An assignment statement “array set a{}” can be used, in case of assigning variables to the entire of an array at one time.

Figure:　Array

### 2-2-3, Arithmetic operation

Integer operation, floating-point operation and comparison of inequality can be executed in Tcl by using expr command.　Operators and mathematical functions are prepared in Tcl as built-in commands.



```
%
(Tcl|Tk) 1 % expr 100+20
120
(Tcl|Tk) 2 % expr 100/20
5
(Tcl|Tk) 3 % expr 100>20
1
(Tcl|Tk) 4 % expr 100<20
0
(Tcl|Tk) 5 % expr rand()
0.333542185991
(Tcl|Tk) 6 % expr sin(1)
0.841470984808
(Tcl|Tk) 7 %
```

expr 100+20　　Return the result of 100+20 expr
100/20　　Return the result of 100/20
expr 100>20　　If the inequality expression evaluates to true, ”1” is returned, and if it evaluates to false, “0” is returned.

Random number function and trigonometric can be also used as arithmetic functions

Figure: Arithmetic operation

The following lists show you operators and functions supported in Tcl, and what they stand for in Tcl.

List: operators

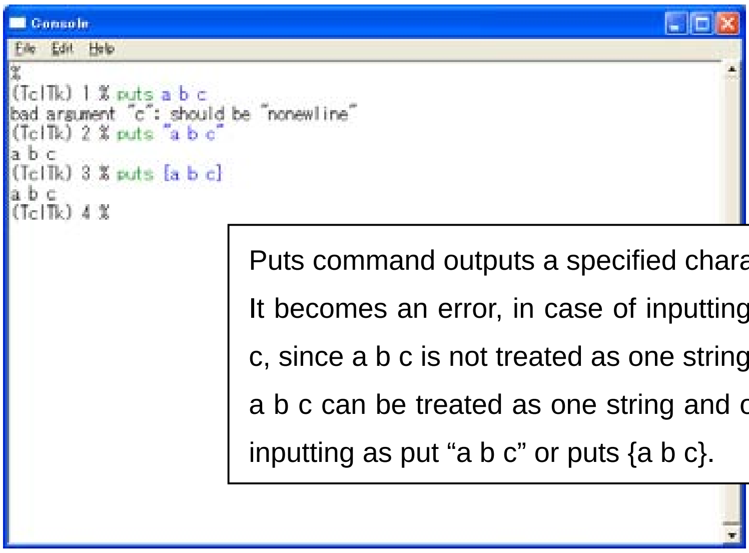
| Signs | Meanings |
|---|---|
| -, +, ^ ,! | Minus sign, Plus sign, complement, negation |
| *, /, % | Multiplication, division, remainder |
| +, - | Addition, subtraction |
| <<, >> | Left-shift, right-shift |
| <, > | comparison in Boolean expression<br>(left-inequality, right-inequality) |
| <=, >= | comparison in Boolean expression<br>(greater than or equal to, less than or equal to) |
| ==, != | Equal sign, inequality sign in Boolean expression |
| Eq, ne | Equality, inequality in Boolean expression (used in strings) |
| &, ^ | Bitwise AND (AND), bitwise exclusive OR (XOR) |
| &&, \|\| | Logical AND, logical OR |
| x?y:z | conditional |

List: functions

| Functions | Meanings |
|---|---|
| Acos, cos, hypot, sinh, asin(), cosh(), log(), sqrt(), atan(), exp(), log10(), tan(), atan2(), floor(), pow(), tanh(), ceil(), fmod(), sin() | Mathematical functions |
| abs(arg) | Absolute value |
| double(arg) | Double-precision value |
| int(arg) | Integer value |
| Rand() | Random number value |
| round(arg) | Round value to integer |
| srand(arg) | Initial value of random number |

### 2-2-4, Double-quotation marks and Curly brackets

An enclosed string within double-quotation marks” “or curly brackets {} is treated as one string in Tcl.



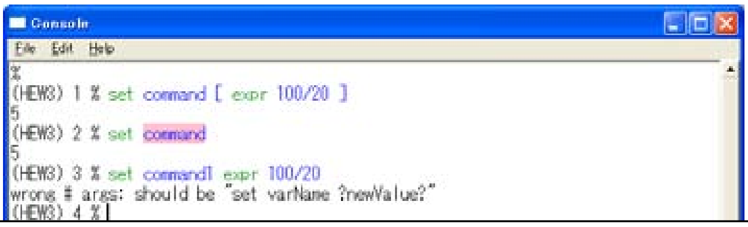
```
%
(TclTk) 1 % puts a b c
bad argument "c": should be "nonewline"
(TclTk) 2 % puts "a b c"
a b c
(TclTk) 3 % puts {a b c}
a b c
(TclTk) 4 %
```

Puts command outputs a specified character.
It becomes an error, in case of inputting puts a b c, since a b c is not treated as one string.
a b c can be treated as one string and output, by inputting as put “a b c” or puts {a b c}.

Figure:  Double-quotation marks and curly brackets

### 2-2-5, Assignment of Commands

An enclosed string within square brackets [] is treated as one command in Tcl.



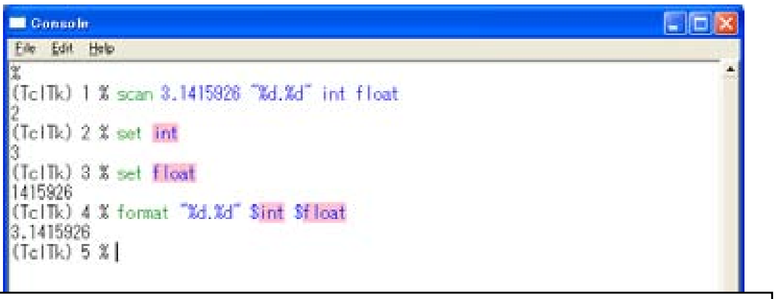
```
%
(HEW3) 1 % set command [ expr 100/20 ]
5
(HEW3) 2 % set command
5
(HEW3) 3 % set command] expr 100/20
wrong # args: should be "set varName ?newValue?"
(HEW3) 4 %
```

An enclosed string within square brackets, [expr 100/20] is treated as one string in Tcl, and set command outputs “5” as the result.
Additionally, expr 100/20 should be assigned to a command since set command is being used.
Therefore, the same result is output by executing set command.
In addition, it becomes an error, in case of not enclosing expr 100/20 within square brackets “[]”.

Figure: Assignment to command

## 2-2-6, Input-Output Format

Scan command and format command are prepared as I/O command in Tcl. These commands are equivalent to scanf and printf in ANSI. Additionally, you can define a style of the format using "%".



```
%
(TclTk) 1 % scan 3.1415926 "%d.%d" int float
2
(TclTk) 2 % set int
3
(TclTk) 3 % set float
1415926
(TclTk) 4 % format "%d.%d" $int $float
3.1415926
(TclTk) 5 %
```

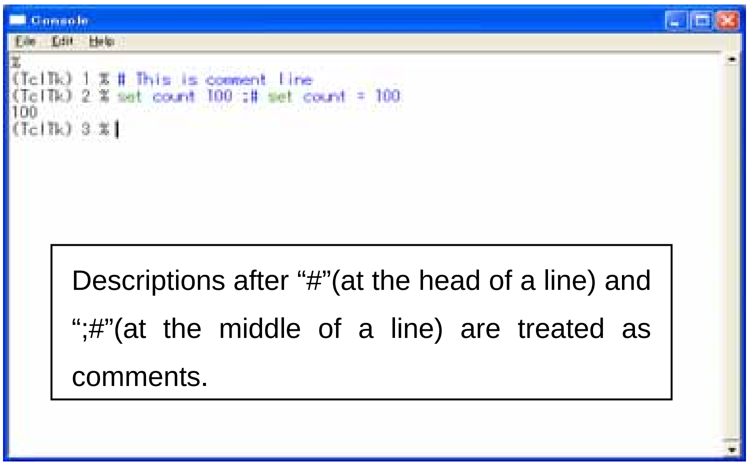
Scan 3.1415926 "%d.%d" int float is equivalent to scanf in ANSI. Scan command scans whole part "3" and fractional part "1415926" of 3.1415926 into int and float variable each. Additionally, it returns 2 (number of times of input values) as the result of the execution

format "%d.%d" $int $float is equivalent to printf in ANSI. The command returns 3.1415926 as the result of an execution.

Figure: Input-output format

## 2-2-7, Description of Comments

In case of describing comments in script files of Tcl, add "#" at the head of a line. Additionally, you will add ";#" at the head of the comments, in case of inserting comments into mid-line.



```
%
(TclTk) 1 % # This is comment line
(TclTk) 2 % set count 100 ;# set count = 100
100
(TclTk) 3 %
```

Descriptions after "#"(at the head of a line) and ";#"(at the middle of a line) are treated as comments.

Figure: Description of comments

### 2-2-8, Procedure

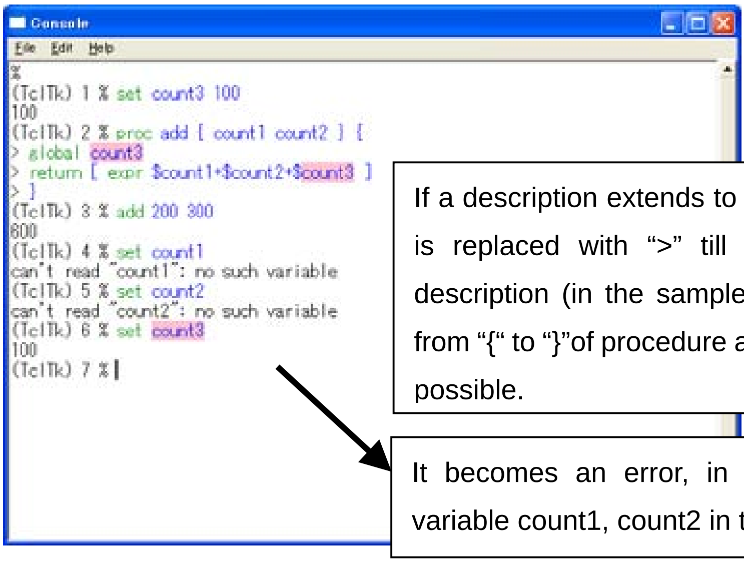
In Tcl, a row of commands can be treated as a function of C language.  A functions called "procedure" can be used as a built-in command in Tcl.

Procedure can be created as a function, which gets 0 or more arguments by using proc command.  Defined variables in procedure are treated as local variables, and can be referred only in the procedure.  Moreover, definition of global variable is required in procedure, in case of referring to global variable in the procedure.

Example of how to create "procedure"

```
~Sample Program~
set count3 100                ;#Assign 100 to variable count3 as an external variable
proc add{ count1 count2 }{    ;#Accept the argument count1 and count2
          global count3       ;#A global definition is required to refer to count3
          return[ expr $count1+$count2+$count3]
                              #Define a value returned by procedure add
                              #using return command
}
add 200 300                   ;#Use procedure add
```

When the sample program is written, procedure add is called, and "600" is returned as the result of "add 200 300".



```
%
(TclTk) 1 % set count3 100
100
(TclTk) 2 % proc add [ count1 count2 ] {
> global count3
> return [ expr $count1+$count2+$count3 ]
> }
(TclTk) 3 % add 200 300
600
(TclTk) 4 % set count1
can't read "count1": no such variable
(TclTk) 5 % set count2
can't read "count2": no such variable
(TclTk) 6 % set count3
100
(TclTk) 7 %
```

If a description extends to a few lines, a prompt is replaced with ">" till the last line of the description (in the sample program, it extends from "{" to "}"of procedure add), and data input is possible.

It becomes an error, in case of referring to variable count1, count2 in the procedure.

Figure: Creation of procedure

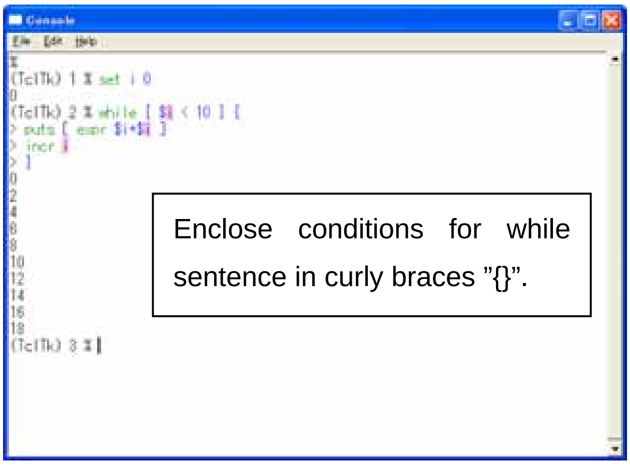
## 2-2-9, Control structure command

Tcl support the essential control structures exist in the other high-level languages.

The control structure commands are while, for, if, if…else, switch, foreach, etc.

① Example of how to use “while”

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

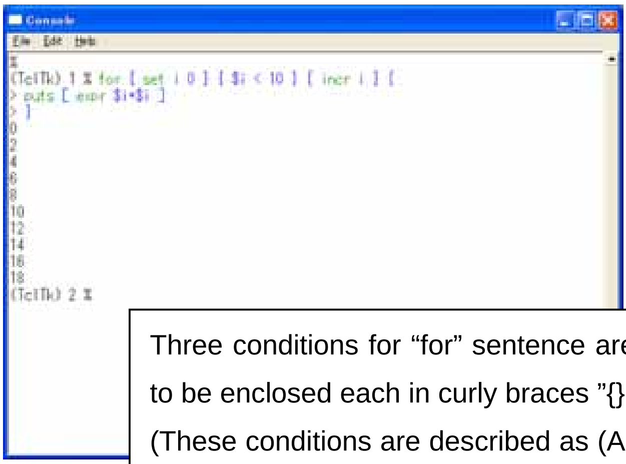


```
%
(TclTk) 1 % set i 0
0
(TclTk) 2 % while { $i < 10 } {
> puts [ expr $i+$i ]
> incr i
> }
0
2
4
6
8
10
12
14
16
18
(TclTk) 3 %
```

Enclose conditions for while sentence in curly braces ”{}”.

② Example of how to use “for”

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

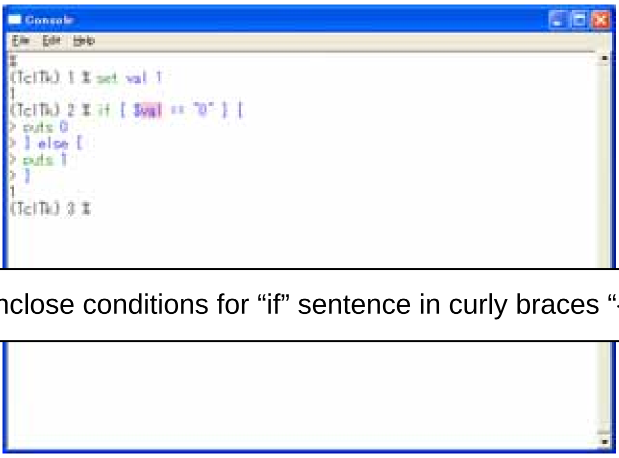


```
%
(TclTk) 1 % for { set i 0 } { $i < 10 } { incr i } {
> puts [ expr $i+$i ]
> }
0
2
4
6
8
10
12
14
16
18
(TclTk) 2 %
```

Three conditions for “for” sentence are required to be enclosed each in curly braces ”{}” in Tcl. (These conditions are described as (A;B;C) in C language.)

③ Example of how to use “if…else”

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

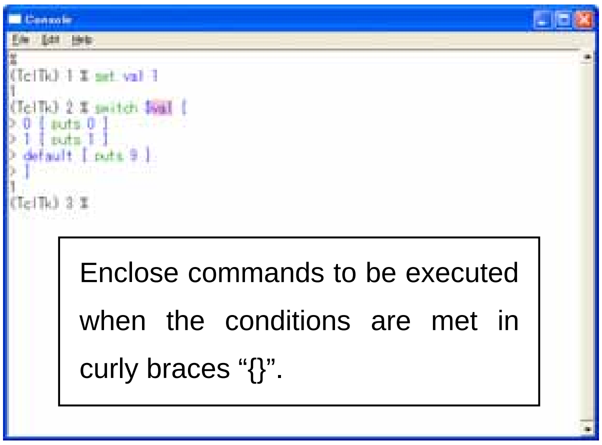


```
%
(Tcl/Tk) 1 % set val 1
1
(Tcl/Tk) 2 % if { $val == "0" } {
> puts 0
> } else {
> puts 1
> }
1
(Tcl/Tk) 3 %
```

Enclose conditions for “if” sentence in curly braces “{}”.

④ Example of how to use “switch”

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

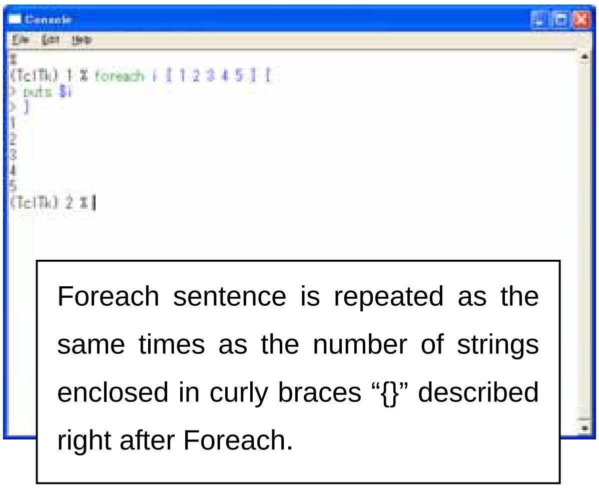


```
%
(Tcl/Tk) 1 % set val 1
1
(Tcl/Tk) 2 % switch $val {
> 0 { puts 0 }
> 1 { puts 1 }
> default { puts 9 }
> }
1
(Tcl/Tk) 3 %
```

Enclose commands to be executed when the conditions are met in curly braces “{}”.

⑤ Example of how to use “foreach”

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

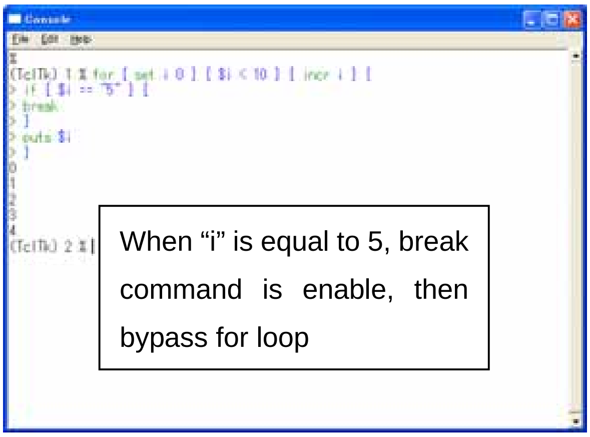


```
%
(TclTk) 1 % foreach i { 1 2 3 4 5 } {
> puts $i
> }
1
2
3
4
5
(TclTk) 2 %
```

Foreach sentence is repeated as the same times as the number of strings enclosed in curly braces “{}” described right after Foreach.

⑥ Continue and break can be used in while, for and foreach sentences.

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



```
%
(TclTk) 1 % for { set i 0 } { $i < 10 } { incr i } {
> if { $i == "5" } {
> break
> }
> puts $i
> }
0
1
2
3
4
(TclTk) 2 %
```

When “i” is equal to 5, break command is enable, then bypass for loop

# 2-3, Programming methods in Tk

## 2-3-1, Basic Grammar

Widget (a supported component of GUI in Tk) is defined and allocated on GUI window, which is newly created by user in Tk programming.
Definition of widget must be described by separating “widget”, “path” and “option” with blank spaces in grammatical rule of Tk programming.

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

A path name must be started with period “.”. “.path” command is generated after an execution of widget command. You can allocate the “.path” command on the newly created GUI window.

```
pack .path
```

“.path” command can be allocated on GUI window by using pack command, which is used to allocate “.path” command.

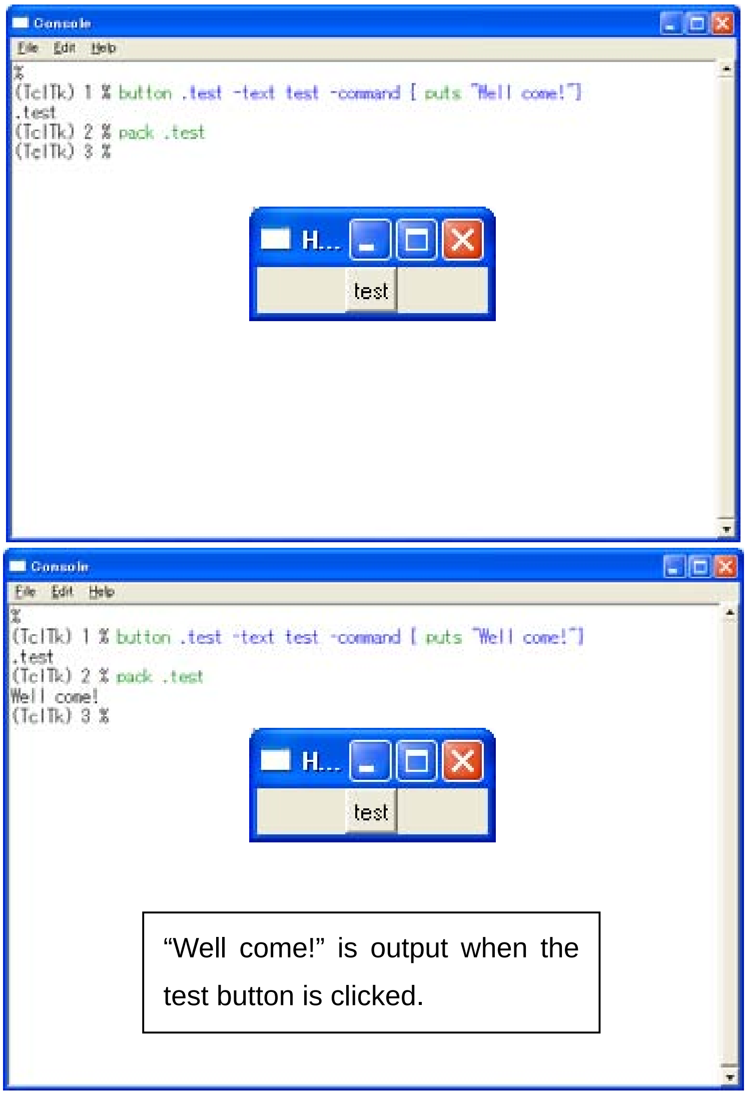
It is possible creating various environments suit the specific needs of users in Tk programming.

# 2-4, Widget in Tk

## 2-4-1 Creation of Button

Button can be created by using button command of Tk.

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



```
%
(TclTk) 1 % button .test -text test -command { puts "Well come!"}
.test
(TclTk) 2 % pack .test
(TclTk) 3 %
```

```
%
(TclTk) 1 % button .test -text test -command { puts "Well come!"}
.test
(TclTk) 2 % pack .test
Well come!
(TclTk) 3 %
```

“Well come!” is output when the test button is clicked.

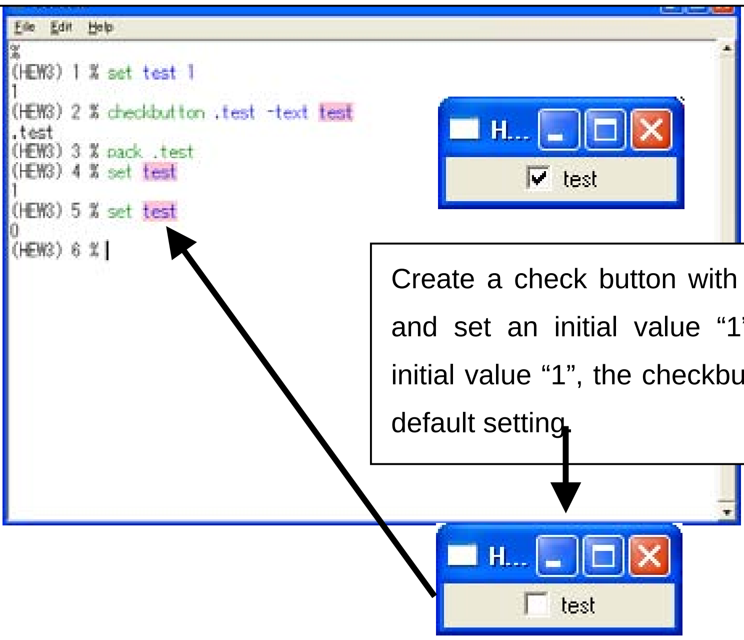
Figure: Creation of button

In the above sample program, a button named “test” is created. Name of button can be specified in -text option. A command to be executed when the button is clicked can be specified in -command option. Therefore, the command enclosed in curly braces “{}” is executed, and output “Well come!” on Console window, when “test” button is clicked.

## 2-4-2, Creation of Check button

Check button can be created by using checkbutton command of Tk.

```
set test 1                              ;#Set “1” to variable “test” as an initial value.
checkbutton .test –text test
pack .test
```



```
%
(HEW3) 1 % set test 1
1
(HEW3) 2 % checkbutton .test -text test
.test
(HEW3) 3 % pack .test
(HEW3) 4 % set test
1
(HEW3) 5 % set test
0
(HEW3) 6 %
```

Create a check button with path name “.test”, and set an initial value “1”. By setting the initial value “1”, the checkbutton is checked by default setting.

When the checkbutton for “test” is cleared, and refer to a value of “test” by set command, it turns to ”0”.

Figure: Creation of checkbutton

In the above sample program, a button named “test” is created. A name of the button can be specified in -text option.

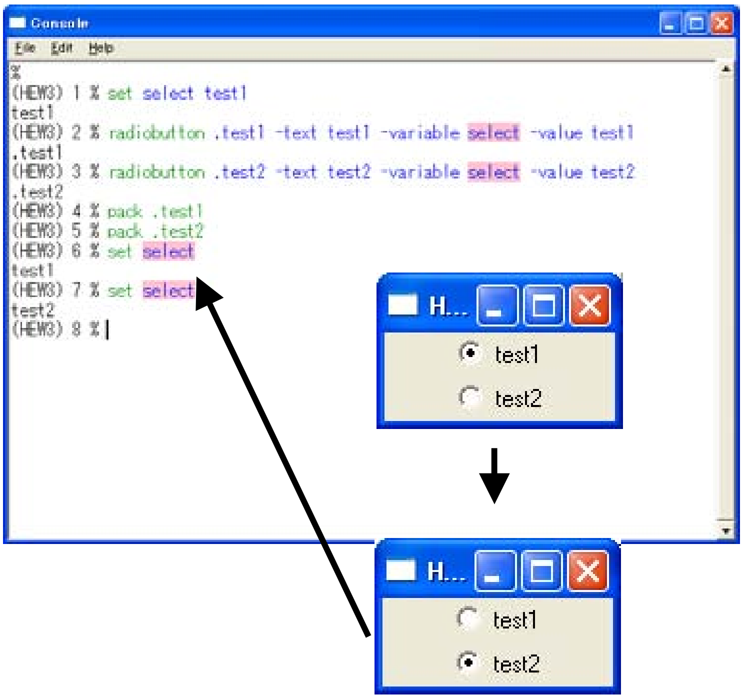
Additionally, the initial value “1”is set to the variable “test” by using set command.

### 2-4-3, Creation of Radiobutton

Radiokbutton can be created by using radiobutton command of Tk.

```
set select test1                  ;#Checkbutton for test 1 is checked by default setting
radiobutton .test1 –text test1 –variable select –valu test1
radiobutton .test2 –text test2 –variable select –vale test2

pack .test1
pack .test2
```



```
%
(HEM3) 1 % set select test1
test1
(HEM3) 2 % radiobutton .test1 -text test1 -variable select -value test1
.test1
(HEM3) 3 % radiobutton .test2 -text test2 -variable select -value test2
.test2
(HEM3) 4 % pack .test1
(HEM3) 5 % pack .test2
(HEM3) 6 % set select
test1
(HEM3) 7 % set select
test2
(HEM3) 8 %
```

When the radiobutton for test2 is clicked, test2 is replaced as a selected value in –variable option.

Figure: Creation of radiobutton

A -value of checked button is passed to a variable select , which is selected in –variable option.　In the above sample program, because “test1” is selected as variable select when the radiobutton is created, the radiobutton “test 1” is checked by default setting.

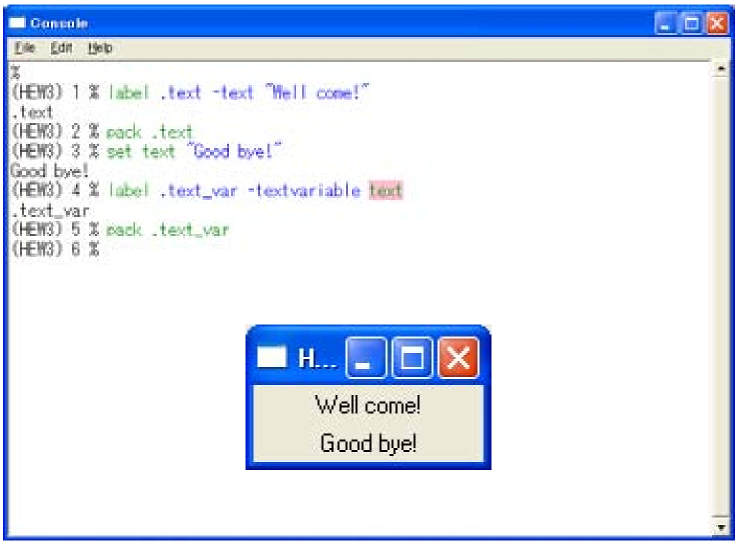
In addition, by naming the variables of multiple radiobuttons specified in –variable option the same, checkboxes are checked exclusively.

## 2-4-4, Creation of Label

A single line message can be displayed on GUI window by using label command of Tk.

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

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



```
%
(HEW3) 1 % label .text -text "Well come!"
.text
(HEW3) 2 % pack .text
(HEW3) 3 % set text "Good bye!"
Good bye!
(HEW3) 4 % label .text_var -textvariable text
.text_var
(HEW3) 5 % pack .text_var
(HEW3) 6 %
```

```
Well come!
Good bye!
```

Figure: Creation of label

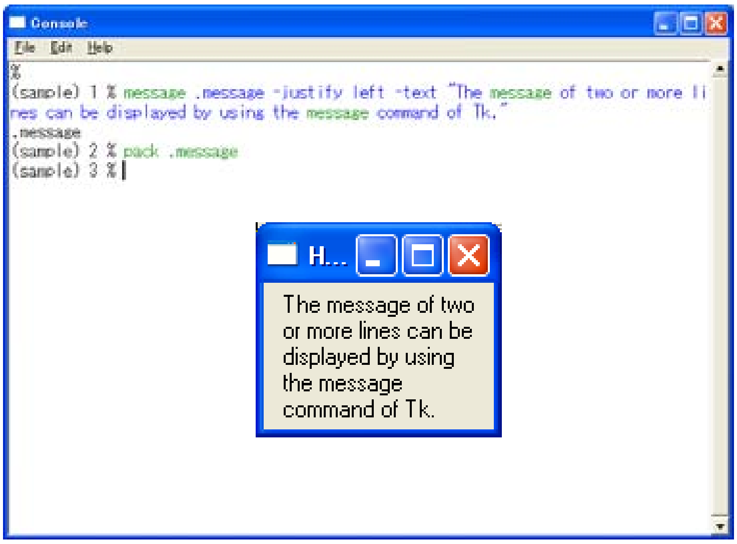
A selected string in -text option of label command is displayed on GUI window.

In addition, it is possible displaying a string, which is set to variable test in advance using –textvariable option.

## 2-4-5, Creation of Message

A few lines message can be displayed on GUI window by using message command of Tk.

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



```
%
(sample) 1 % message .message -justify left -text "The message of two or more li
nes can be displayed by using the message command of Tk."
.message
(sample) 2 % pack .message
(sample) 3 %
```

The message of two
or more lines can be
displayed by using
the message
command of Tk.

Figure: Creation of message

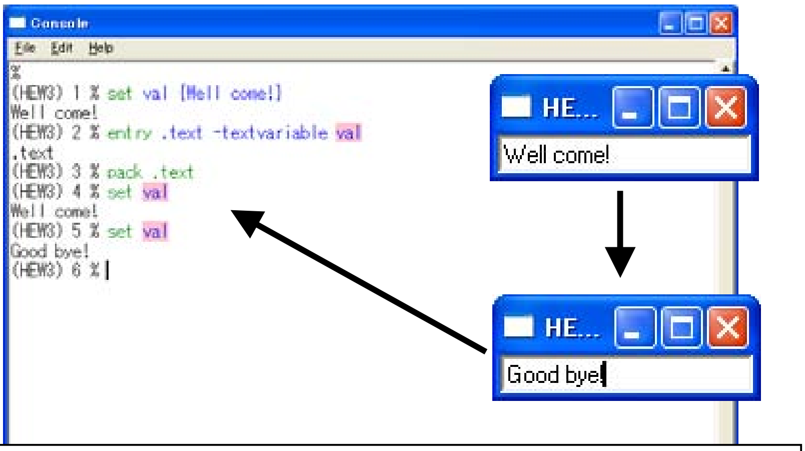
A few lines message can be displayed by using message command, although only a single line message can be displayed by using label command.

In addition, in case of that a command message is too long to input on a line, you can start a new line using “¥”.

### 2-4-6, Creation of Entry

Output dialog can be created by using entry command of Tk.

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



```
%
(HEW3) 1 % set val {Well come!}
Well come!
(HEW3) 2 % entry .text -textvariable val
.text
(HEW3) 3 % pack .text
(HEW3) 4 % set val
Well come!
(HEW3) 5 % set val
Good bye!
(HEW3) 6 %
```

Well come!

Good bye!

"Well come!" is set as an initial value of variable val by using set command. Therefore, "Well come!" is displayed by default, when the input dialog is created.

When "Good bye!" is input in the input dialog, the input string "Good bye!" is reflected in variable val.

Figure: Creation of entry

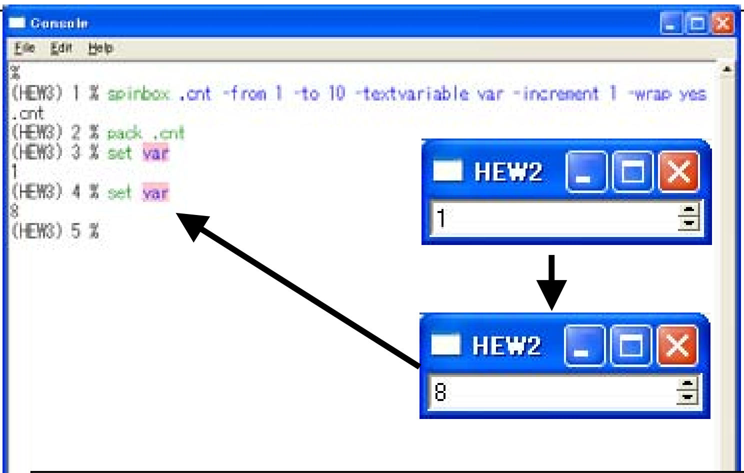
Input dialog created using entry command can accept a single line string and no more. The input string is reflected in variable val, which is specified in -textvariable option.

In case of creating an input dialog, which can accept more than a single line string, you can use text command prepared in Tk. In addition, text command offers you a diversity of options. By using those options, you can create simple editor also.

## 2-4-7, Creation of Spinbox

Spinbox with scroll bar can be created by using spinbox command of Tk

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



```
%
(HEW3) 1 % spinbox .cnt -from 1 -to 10 -textvariable var -increment 1 -wrap yes
.cnt
(HEW3) 2 % pack .cnt
(HEW3) 3 % set var
1
(HEW3) 4 % set var
8
(HEW3) 5 %
```

The value can be changed using the scroll bar. The specified value is reflected in variable val specified in –textvariable.

Figure: Ceation of spinbox

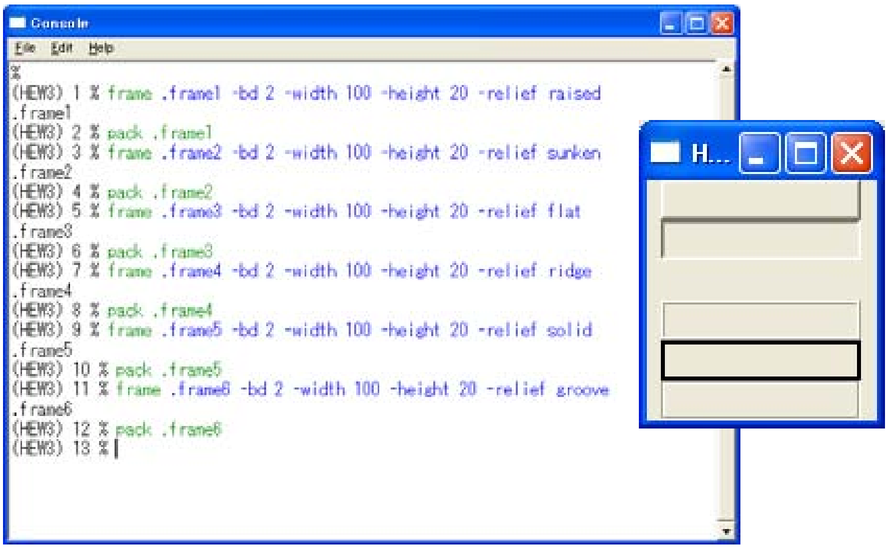
You can specify the range of the settable values in the spinbox using –from and –to options of spinbox command. You can also specify the interval between the values, which is displayed in the spinbox using –increment option. The specified value is reflected in variable val specified in –textvariable.

Additionally, in case of selecting “yes” in –wrap option, the value circulates among the range of the values you specified in –from and –to options. (For example; 0 ~..~ 10 ~ 0 ~..~ 5 )

### 2-4-8, Creation of Frame

Frame can be created by using frame command of Tk.

```
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
```



```
%
(HEW3) 1 % frame .frame1 -bd 2 -width 100 -height 20 -relief raised
.frame1
(HEW3) 2 % pack .frame1
(HEW3) 3 % frame .frame2 -bd 2 -width 100 -height 20 -relief sunken
.frame2
(HEW3) 4 % pack .frame2
(HEW3) 5 % frame .frame3 -bd 2 -width 100 -height 20 -relief flat
.frame3
(HEW3) 6 % pack .frame3
(HEW3) 7 % frame .frame4 -bd 2 -width 100 -height 20 -relief ridge
.frame4
(HEW3) 8 % pack .frame4
(HEW3) 9 % frame .frame5 -bd 2 -width 100 -height 20 -relief solid
.frame5
(HEW3) 10 % pack .frame5
(HEW3) 11 % frame .frame6 -bd 2 -width 100 -height 20 -relief groove
.frame6
(HEW3) 12 % pack .frame6
(HEW3) 13 %
```

Figure: Creation of frame

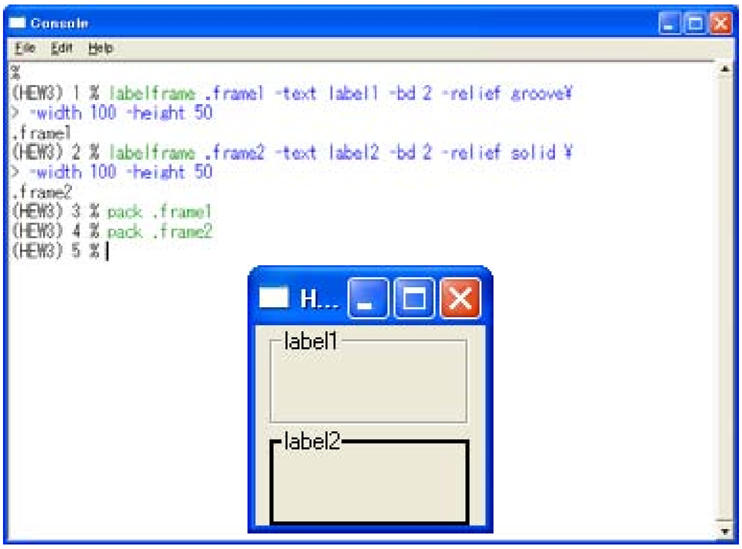
A newly created GUI window by user can be arranged using frame command. The allocation method of widget to frames is described in a later chapter.

You can specify the size of a frame in –width and –height options of frame command. You can also change the shape of a frame in relief option.

### 2-4-9, Creation of Labelframe

Labelframe can be created using labelframe command of Tk.

```
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
```



```
%
(HEW3) 1 % labelframe .frame1 -text label1 -bd 2 -relief groove¥
> -width 100 -height 50
.frame1
(HEW3) 2 % labelframe .frame2 -text label2 -bd 2 -relief solid ¥
> -width 100 -height 50
.frame2
(HEW3) 3 % pack .frame1
(HEW3) 4 % pack .frame2
(HEW3) 5 %
```

Figure: Creation of labelframe

A newly created GUI window by user can be arranged using labelframe command as well as frame command. The difference of frame command is that you can name the frames using –text option.

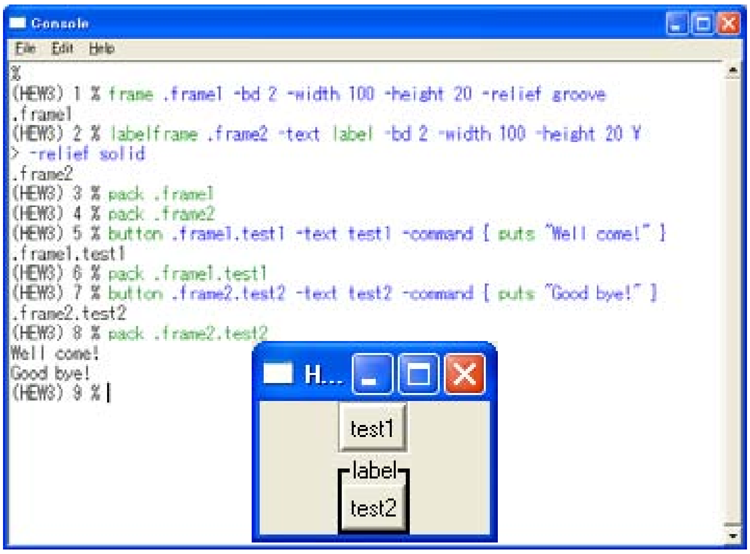
In addition, you can specify the size of a frame in –width and –height options of labelframe command as well as frame command. You can also specify the shape of a frame in relief option of labelframe command as well as frame command.

## 2-4-10, Allocation of widget to frame/labelframe

You can allocate widget of Tk to frame/labelframe created in Chapter 2-4-9,2-4-10.

```
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
```



```
%
(HEW3) 1 % frame .frame1 -bd 2 -width 100 -height 20 -relief groove
.frame1
(HEW3) 2 % labelframe .frame2 -text label -bd 2 -width 100 -height 20 ¥
> -relief solid
.frame2
(HEW3) 3 % pack .frame1
(HEW3) 4 % pack .frame2
(HEW3) 5 % button .frame1.test1 -text test1 -command [ puts "Well come!" ]
.frame1.test1
(HEW3) 6 % pack .frame1.test1
(HEW3) 7 % button .frame2.test2 -text test2 -command [ puts "Good bye!" ]
.frame2.test2
(HEW3) 8 % pack .frame2.test2
Well come!
Good bye!
(HEW3) 9 %
```

Figure: Example of widget allocation

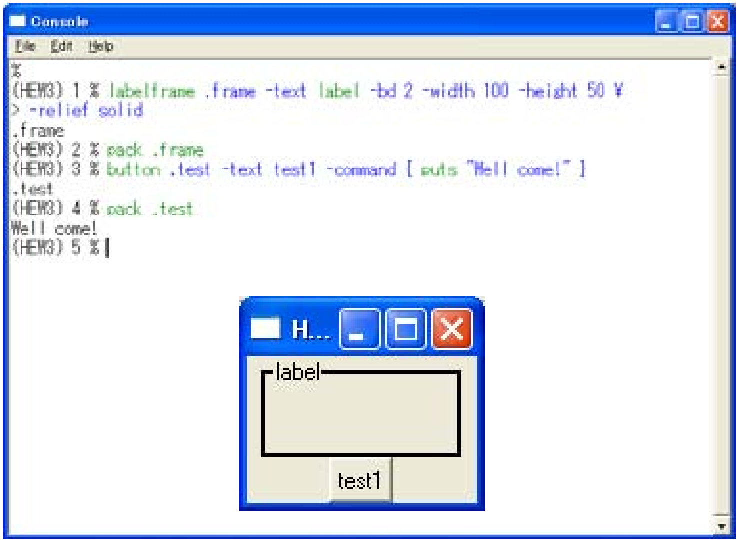
In case of allocating widget of Tk to frame or labelframe, the path name is different from ordinary.

In the above sample program, buttons are allocated to the frame and the labelframe. The path name of the frame is “.frame1”, and the path name of the labelframe is “.frame2”. Therefore, the path names of allocated buttons to the frame and the labelframe are specified as “.frame1.test1” and “.frame2.test2” each. These path names mean that the button “.test1” is allocated to the frame “.frame1”, and the button “.test2” is allocated to the labelframe “.frame2”.

Additionally, in case of not using the path names of frame or labelframe but the path name of widget of Tk, widget is allocated outside of the frame or the labelframe.

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

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



```
%
(HEM3) 1 % labelframe .frame -text label -bd 2 -width 100 -height 50 ¥
> -relief solid
.frame
(HEM3) 2 % pack .frame
(HEM3) 3 % button .test -text test1 -command [ puts "Well come!" ]
.test
(HEM3) 4 % pack .test
Well come!
(HEM3) 5 %
```

Figure: Example of widget allocation

-----------------------------------------------------------------------------------------------------------------------

~Supplementary information~

“.(period)” used in path name of Tcl/Tk shows the route. By allocating frames and labelframes to the route, the path names in GUI environment shows the layered system

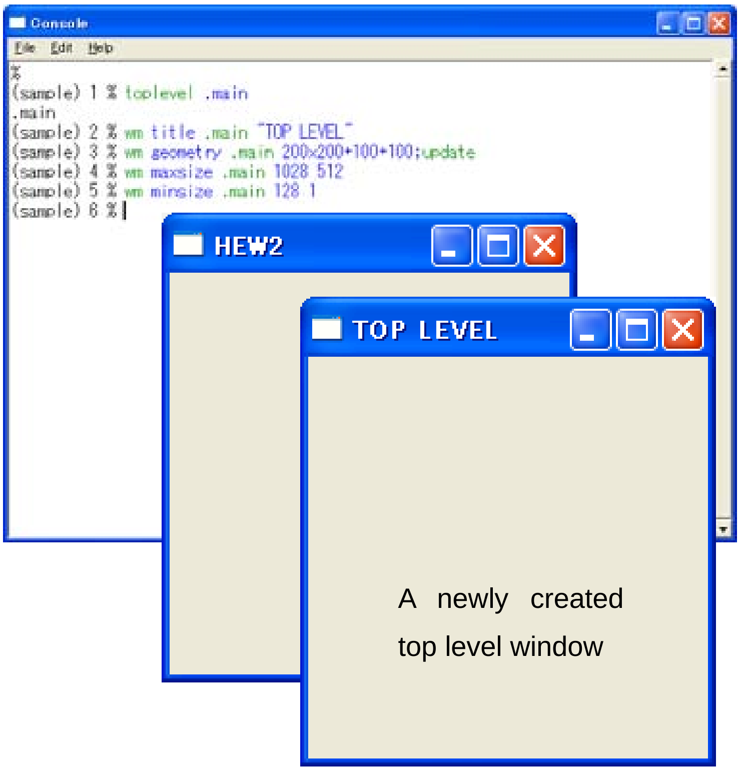
For example:

| | | |
|---|---|---|
| Path name | .top | top of GUI environment |
| | .top.frame1 | a path name of frame1 allocated on top |
| | .top.frame2 | a path name of frame2 allocated on top |
| | .top.frame2.subframe | a path name of subframe allocated on frame3 |

## 2-4-11, Creation of New Top Level Window

Top level window can be created newly using toplevel command of Tk.

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



```
%
(sample) 1 % toplevel .main
.main
(sample) 2 % wm title .main "TOP LEVEL"
(sample) 3 % wm geometry .main 200x200+100+100;update
(sample) 4 % wm maxsize .main 1028 512
(sample) 5 % wm minsize .main 128 1
(sample) 6 %
```

A newly created top level window

Figure: Creation of top level window

In the above sample program, a window is created newly besides the window prepared in HEW by default by using toplevel command. The title and size of the new window can be specified using wm command.

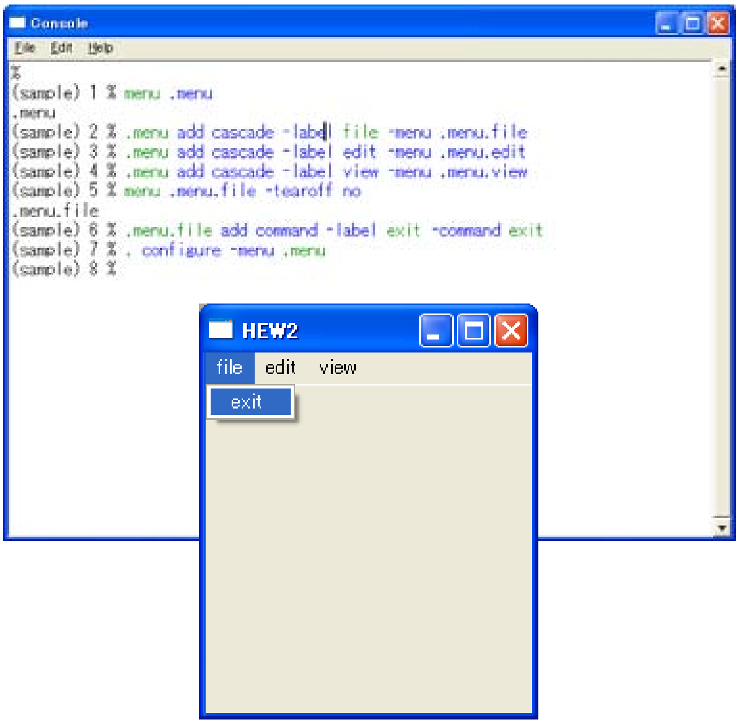
In case of allocating widget of Tk on the new window, the path name must be started with “.main”.

Additionally, the new window can be deleted by destroy command (destroy .path name).

### 2-4-12, Creation of Menu

Tool menu can be specified by using menu command of Tk.

```
menu .menu          ;#Specify a path name for menu
.menu add cascade –label file –menu .menu.file
.menu add cascade –label edit –menu .menu.edit
.menu add cascade –label view –menu .menu.view
                    #Specify cascades (file, edit, view) add to .menu
menu .menu.file –tearoff no
                    #When you select “yes” for –tearoff option, the created menu can be
                    #deleted from the window
.menu.file add command –label exit –command exit
                    #Allocate “exit” to menu.file as a submenu
                    #Define “exit” as a submenu for when ”file” is selected
. configure –menu .menu
```



```
%
(sample) 1 % menu .menu
.menu
(sample) 2 % .menu add cascade -label file -menu .menu.file
(sample) 3 % .menu add cascade -label edit -menu .menu.edit
(sample) 4 % .menu add cascade -label view -menu .menu.view
(sample) 5 % menu .menu.file -tearoff no
.menu.file
(sample) 6 % .menu.file add command -label exit -command exit
(sample) 7 % . configure -menu .menu
(sample) 8 %
```

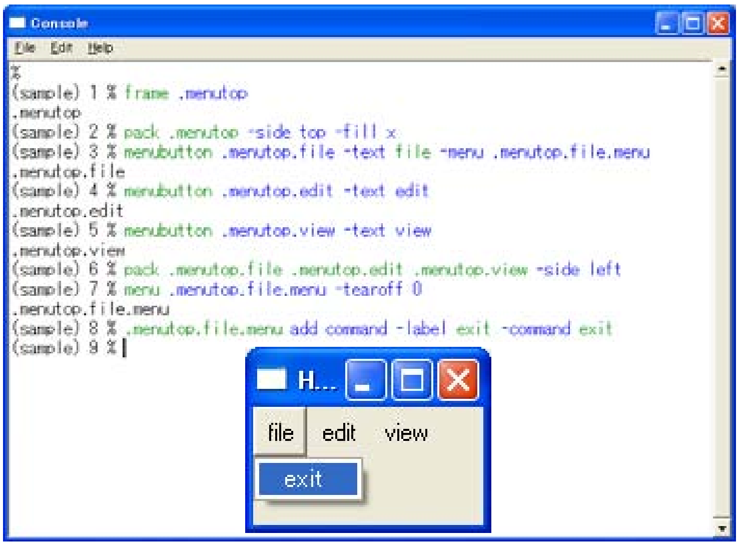
Figure: Creation of tool menu

Pull-down menu can be created on toplevel window using menu command.

## 2-4-13, Creation of Menu button

Menu button can be created using menubutton command of Tk.

```
frame .menutop                 ;#Use frame command to allocate a frame on the tool bar
pack .menutop –side top –fill x
                               #The created window in -slide top is allocated to top
menubutton .menutop.file –text file –menu .menutop.file.menu
                               #Allocate .menutop.file.menu as a submenu in –menu option
menubutton .menutop.edit –text edit
menubutton .menutop.view –text view
pack .menutop.file .menutop.edit .menutop.view –side left
                               #Allocate the created menubuttons
menu .menutop.file.menu –tearoff 0
                               # When selecting “true” in -tearoff option, the created menu can
                               #be deleted from the window
.menutop.file.menu add command –label exit –command exit
                               #Define an a submenu for when ”menutop.file.menu” is selected
```



```
%
(sample) 1 % frame .menutop
.menutop
(sample) 2 % pack .menutop -side top -fill x
(sample) 3 % menubutton .menutop.file -text file -menu .menutop.file.menu
.menutop.file
(sample) 4 % menubutton .menutop.edit -text edit
.menutop.edit
(sample) 5 % menubutton .menutop.view -text view
.menutop.view
(sample) 6 % pack .menutop.file .menutop.edit .menutop.view -side left
(sample) 7 % menu .menutop.file.menu -tearoff 0
.menutop.file.menu
(sample) 8 % .menutop.file.menu add command -label exit -command exit
(sample) 9 %
```

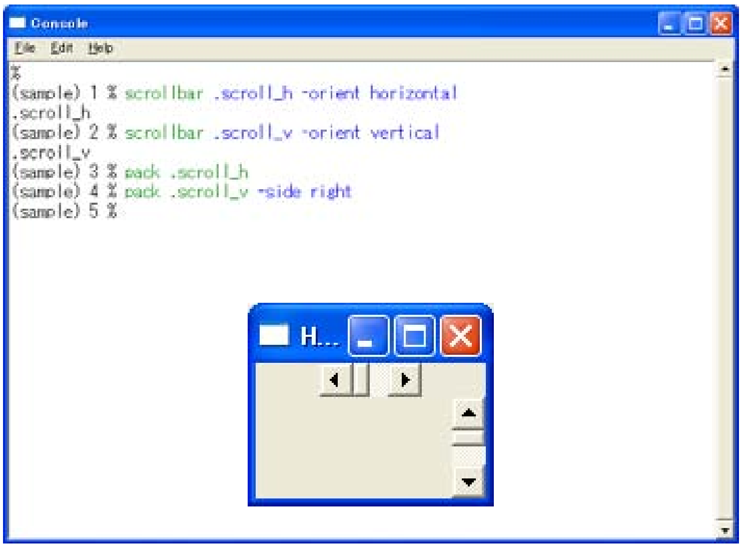
Figure: Creation of menu button

Created menu by menubutton command is the functionally same as one created by menu command. In case of allocating the menu on tool bar, a frame for the menu bar must be allocated in advance using frame command.

## 2-4-14, Creation of Scroll bar

Scroll bar can be created on windows, etc. using scrollbar command of Tk.

```
scrollbar .scroll_h –orient horizontal          ;#Define a horizontal scroll bar
scrollbar .scroll_v –orient vertical            ;# Define a vertical scroll bar
pack .scroll_h                                  ;#Allocate the defined scroll bars
pack .scroll_v –side right
```



```
%
(sample) 1 % scrollbar .scroll_h -orient horizontal
.scroll_h
(sample) 2 % scrollbar .scroll_v -orient vertical
.scroll_v
(sample) 3 % pack .scroll_h
(sample) 4 % pack .scroll_v -side right
(sample) 5 %
```

Figure: Creation of scroll bars

In the above sample program, the scroll bars are allocated on the top window. Scroll bar also can be allocated on frame, etc., by creating a frame in advance and specifying the path name for the scroll bar at the time of definition and allocation.

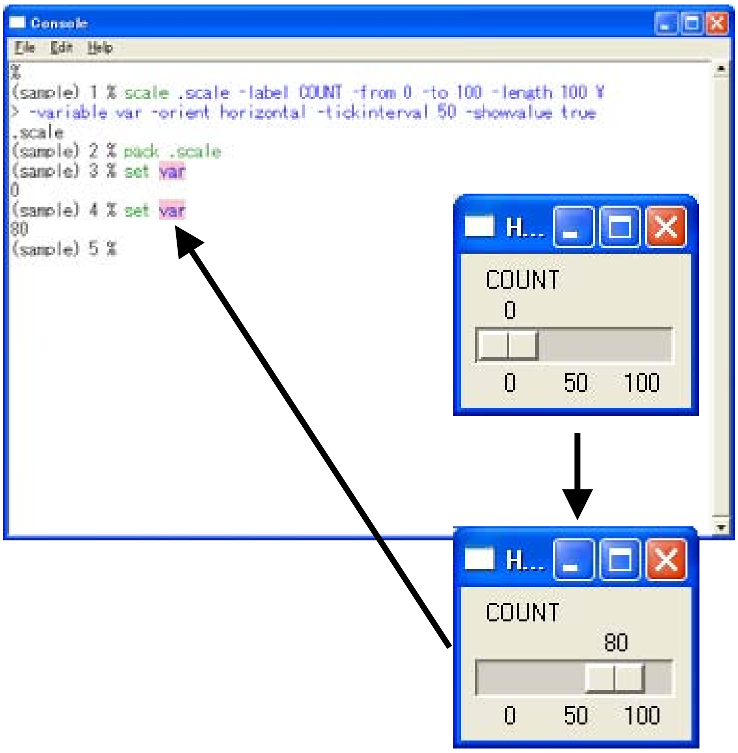
```
labelframe .frame –text label –bd 2 –width 100 –height 50 –relief solid
pack .frame

scrollbar .frame.scroll_v –orient vertical     ;#Specify the path name of the frame
pack .frame.scroll_v
```

## 2-4-15, Creation of Scale bar to change the variable value

Scale bar to change a value of variable can be created using scale command of Tk.

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



```
%
(sample) 1 % scale .scale -label COUNT -from 0 -to 100 -length 100 ¥
> -variable var -orient horizontal -tickinterval 50 -showvalue true
.scale
(sample) 2 % pack .scale
(sample) 3 % set var
0
(sample) 4 % set var
80
(sample) 5 %
```

When the value pointed on the scale bar is changed, it is reflected in a value of variable val, which is specified in -variable option.

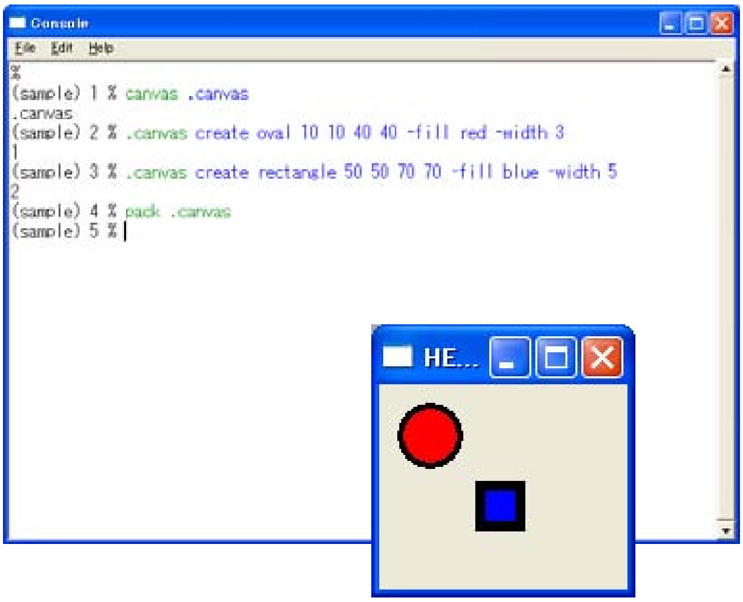
Figure: Creation of scale bar

You can specify the range of the value on the scale bar using -from and -to options of scale command. The indicated value on the scale bar is reflected in variable val, which is specified in -variable option. Additionally, you can specify the interval of indicator scale to be displayed on the scale bar in –tickinterval option, and also a display value, which is indicated on the scale bar in –showvale option.

### 2-4-16, Creation of Canvas

Canvas with lines, texts and polygons can be created using canvas command of Tk.

```
canvas .canvas                          ;#Define a canvas you create
.canvas create oval 10 10 40 40 –fill red –width 3
.canvas create rectangle 50 50 70 70 –fill blue –width 5
pack .canvas                            ;#Allocate the canvas
```



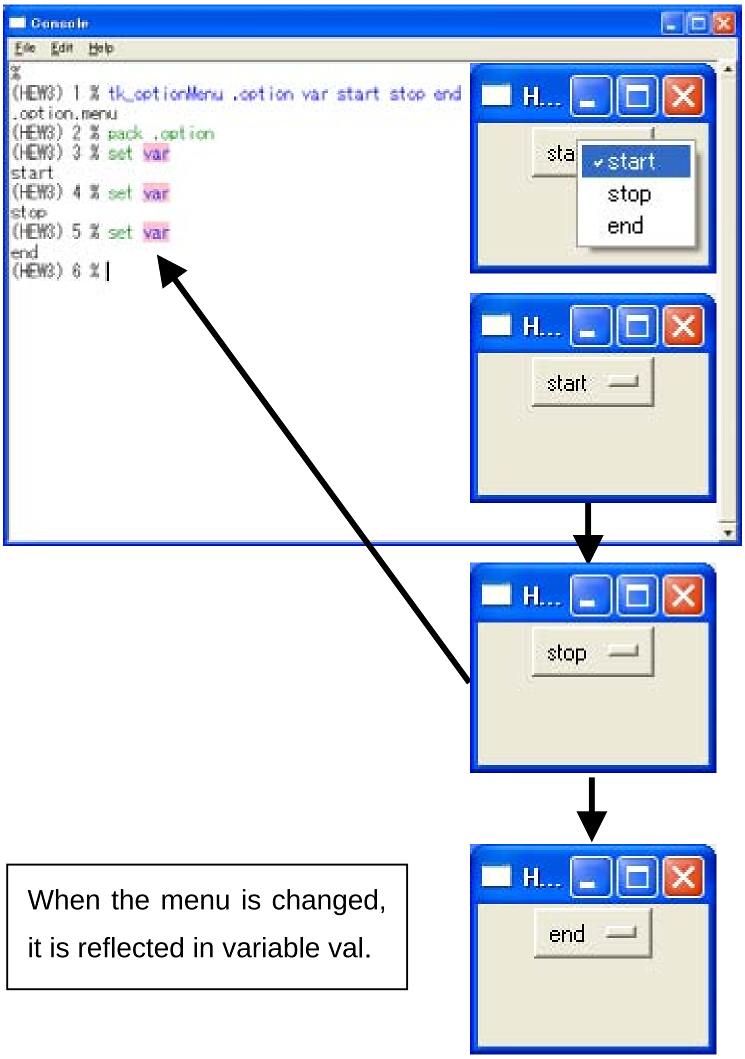
```
%
(sample) 1 % canvas .canvas
.canvas
(sample) 2 % .canvas create oval 10 10 40 40 -fill red -width 3
1
(sample) 3 % .canvas create rectangle 50 50 70 70 -fill blue -width 5
2
(sample) 4 % pack .canvas
(sample) 5 %
```

Figure: Creation of canvas

## 2-4-17, Creation of Option menu

Option menu can be created using tk_optionMenu command of Tk.

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



```
%
(HEW3) 1 % tk_optionMenu .option var start stop end
.option.menu
(HEW3) 2 % pack .option
(HEW3) 3 % set var
start
(HEW3) 4 % set var
stop
(HEW3) 5 % set var
end
(HEW3) 6 %
```

When the menu is changed, it is reflected in variable val.

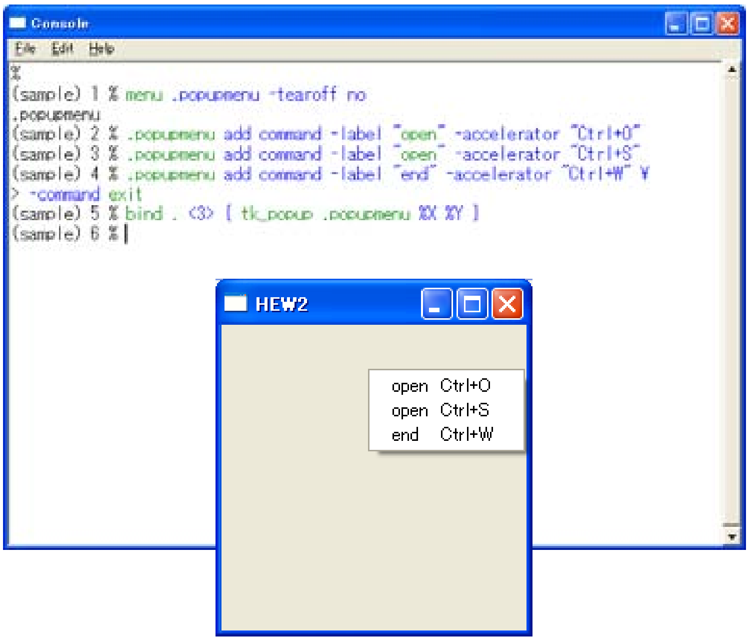
Figure: Creation of option menu

In the above sample program, variable val and the option menus (“start”, “stop”, “end”) are specified using tk_option command. Variable val reflects a specified menu among the three menus.

## 2-4-18, Creation of Pop-up menu

Pop-up menu can be created using tk_popup command of Tk.

```
menu .popupmenu –tearoff no
          #The pop-up menu can be deleted from the window by selecting “yes” for
          #–tearoff option
.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
          #A submenu for when “end” is selected can be specified in –command option
bind . <3> { tk_popup .popupmenu %X %Y }
```



```
%
(sample) 1 % menu .popupmenu -tearoff no
.popupmenu
(sample) 2 % .popupmenu add command -label "open" -accelerator "Ctrl+O"
(sample) 3 % .popupmenu add command -label "open" -accelerator "Ctrl+S"
(sample) 4 % .popupmenu add command -label "end" -accelerator "Ctrl+W" ¥
> -command exit
(sample) 5 % bind . <3> { tk_popup .popupmenu %X %Y }
(sample) 6 %
```

Figure: Creation of pop-up menu

In the above sample program, the created “.popupmenu” is defined as a pop-up menu using tk_popup command. You can specify the location of the pop-up window to be opened on the window by setting the value for %X and %Y in the following description.

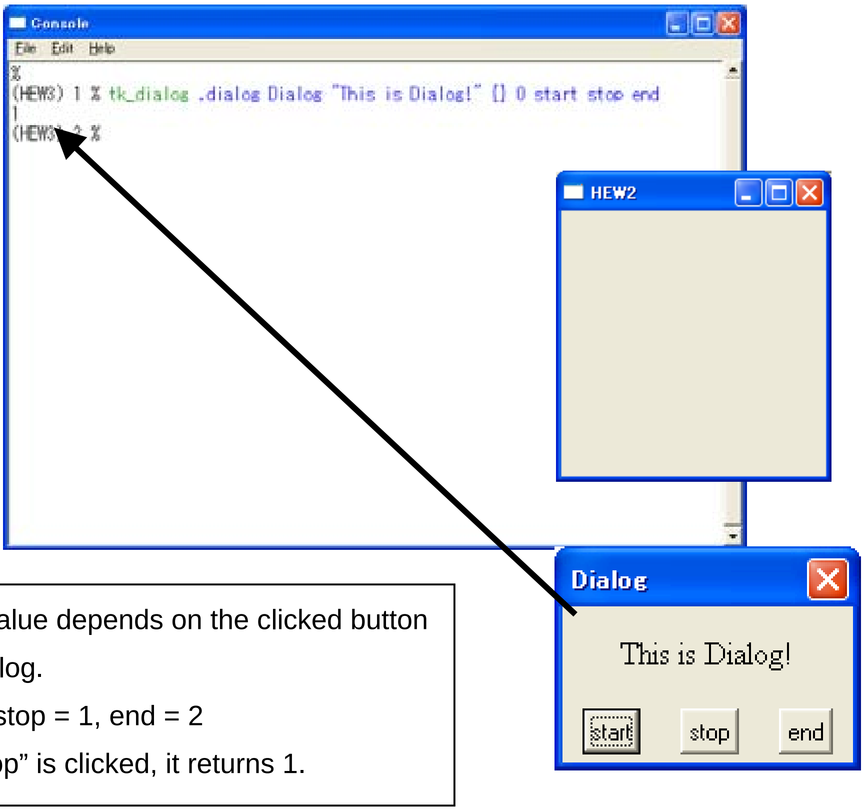
```
          bind . <3> { tk_popup .popupmenu %X %Y }
```

Therefore the pop-up menu can be opened anywhere on the window.

## 2-4-19, Creation of Simple dialog

Simple dialog can be created using tk_dialog command of Tk.

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



```
%
(HEW3) 1 % tk_dialog .dialog Dialog "This is Dialog!" {} 0 start stop end
1
(HEW3) 2 %
```

This is Dialog!

A return value depends on the clicked button on the dialog.

start = 0, stop = 1, end = 2

When “stop” is clicked, it returns 1.

Figure: Creation of dialog

You can specify a windows title and a message on a dialog box using tk_dialog command. In the above sample dialog, the window is titled “Dialog”, and the message is specified as “This is Dialog!”.

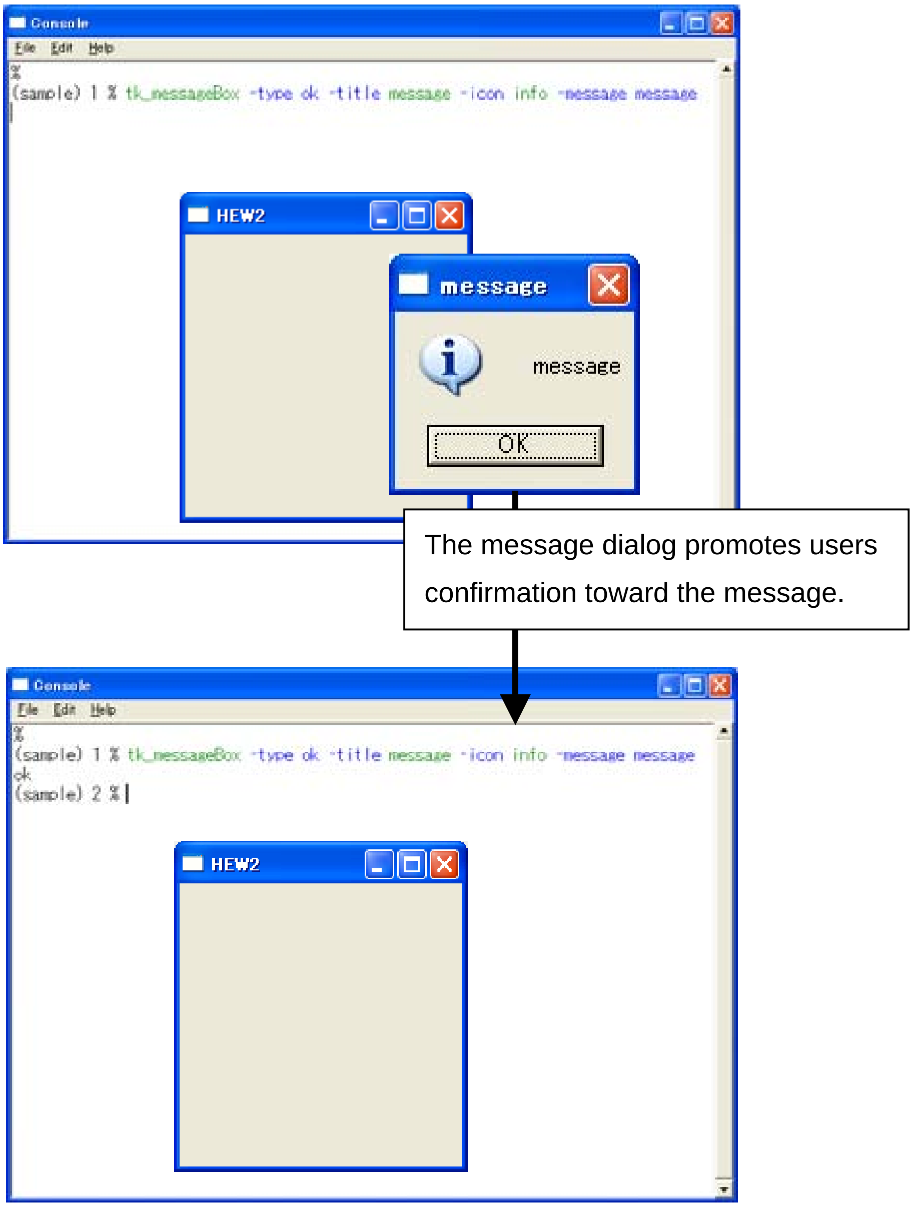
You can also specify names of buttons on the dialog box and their initial values. In the above sample dialog, the buttons are named “start”, “stop” and “end”, and their initial values are specified as “0”, “1” and “2” each.

One of the return values “0”, “1” or “2” is returned as the result by clicking one of the buttons.

## 2-4-20, Creation of Message dialog

Message dialog can be created using tk_messageBox command of Tk.

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



```
%
(sample) 1 % tk_messageBox -type ok -title message -icon info -message message
```

The message dialog promotes users confirmation toward the message.

```
%
(sample) 1 % tk_messageBox -type ok -title message -icon info -message message
ok
(sample) 2 %
```

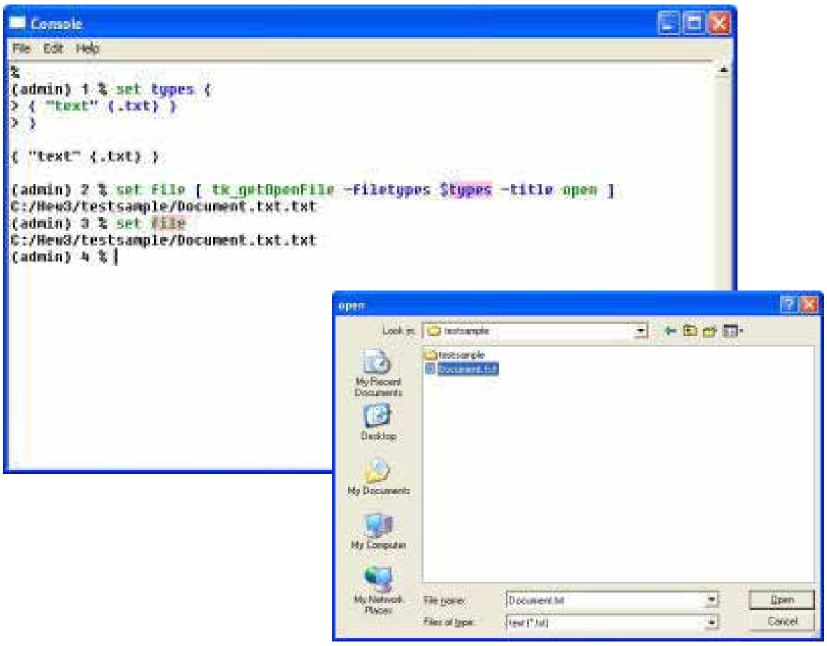
Figure: Creation of message dialog

You can create interactive message dialog box by using tk_messageBox command. You can specify a type of the button in –type option, a title of the button in –title option, and also a message to be displayed on the dialog in –message option.

## 2-4-21, Creation of Exist File Open dialog

Dialog to open existing files can be created using tk_getOpenFile command of Tk.

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



```
%
(admin) 1 % set types {
> { "text" {.txt} }
> }

{ "text" {.txt} }

(admin) 2 % set file [ tk_getOpenFile -filetypes $types -title open ]
C:/Hew3/testsample/Document.txt.txt
(admin) 3 % set file
C:/Hew3/testsample/Document.txt.txt
(admin) 4 %
```

File name: Document.txt
Files of type: text (*.txt)

When an existing file is selected and opened, the storage location of the file is reflected in file variable.

Figure: Creation of file open dialog

You can specify a file type in –filetypes option, and a window title in –title option, when creating a file open dialog using Tk_getOpenFile command.  In the above program, type variable is selected as the file type in –filetypes option.  text [ *.txt] is specified as the file type for type variable by set type{ …} command in advance.

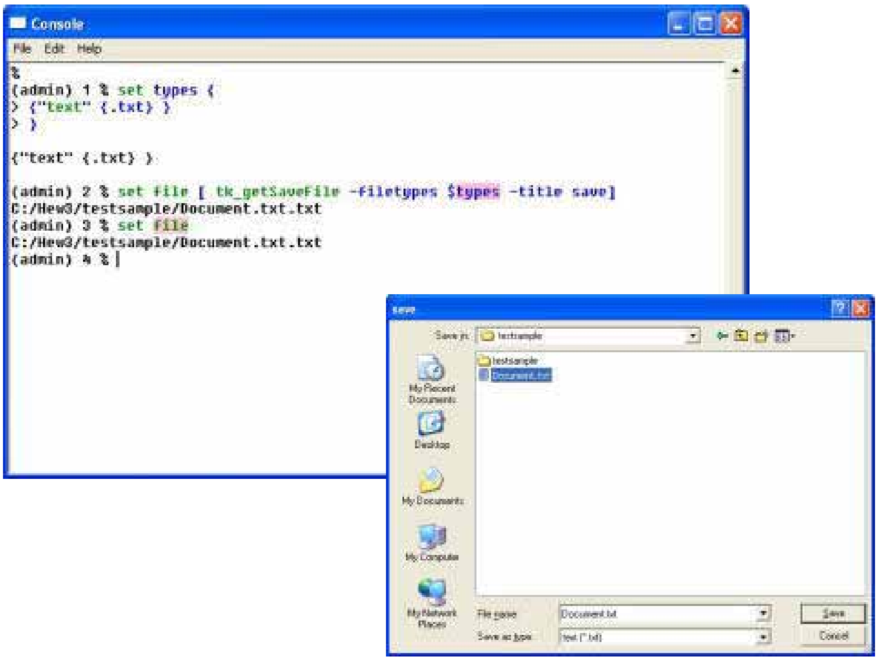
In addition, a storage location of the selected file is returned as the result of Tk_getOpenFile command.

In the above program, the result is stored in file variable using set command.

## 2-4-22, Creation of New File Open dialog

Dialog to open and save new file can be created using tk_getSaveFile command of Tk.

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



```
%
(admin) 1 % set types {
> {"text" {.txt} }
> }

{"text" {.txt} }

(admin) 2 % set file [ tk_getSaveFile -filetypes $types -title save]
C:/Hew3/testsample/Document.txt.txt
(admin) 3 % set file
C:/Hew3/testsample/Document.txt.txt
(admin) 4 % |
```

When a new file is selected and saved, a storage location of the saved file is reflected in file variable.

Figure: Creation of file save dialog

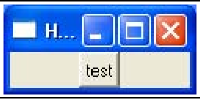
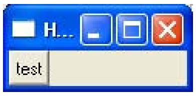
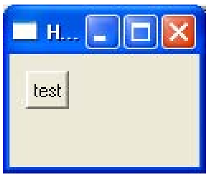
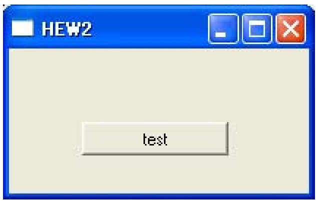
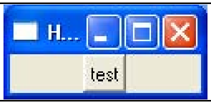
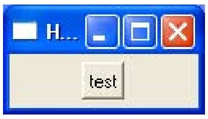
You can specify a file type in –filetypes option, and a window title in –title option, when creating a file open dialog using Tk_getSaveFile command.

Additionally, a storage location of the selected file is returned as the result of Tk_getSaveFile command.

## 2-4-23, Allocation of widget

In the previous chapters, the allocation method of widget is described only by using pack command. However, you can also use place command and grid command to allocate widget in addition to pack command. The allocation method varies by command, so you can choose the most appropriate command depending on GUI window you want to create.

Example of widget allocation

<table>
<tr><th>Command</th><th>Example of use of command</th><th>Allocation example</th></tr>
<tr><td rowspan="3">pack</td><td colspan="2">Lay out widget by direction.<br>The direction can be set in -side [left, right, top, bottom] option, etc..</td></tr>
<tr><td>pack [ button .test –text test –command { go } ]</td><td> </td></tr>
<tr><td>pack [ button .test –text test –command { go } ] –side left</td><td> </td></tr>
<tr><td rowspan="3">place</td><td colspan="2">Lay out widget by coordinates.<br>The coordinates can be specified in –x and –y options, and the size can be specified in –width option.</td></tr>
<tr><td>button .test –text test –command { go }<br>place .test –x 10 –y 10</td><td> </td></tr>
<tr><td>button .test –text test –command { go }<br>place .test –x 50 –y 50 –width 100</td><td> </td></tr>
<tr><td rowspan="3">grid</td><td colspan="2">Lay out widget as grid.<br>The grid location can be specified in –column and –row options, and the margin can be specified in –padx and –pady options.</td></tr>
<tr><td>button .test –text test –command { go }<br>grid .test</td><td> </td></tr>
<tr><td>button .test –text test –command { go }<br>grid .test –column 3 –row 4 -padx 3 –pady 5</td><td> </td></tr>
</table>

Options used in the above list are only some of examples among many. You can use more options. Please refer to Tcl/Tk reference manual, etc., for further information.

# 3, Tcl/Tk programming in HEW

Tcl/Tk programming in HEW is possible.

Tcl/Tk create development environments, which suit the needs of users by allocating HEW commands to GUI window (buttons, etc.) created in Tcl/Tk.

Therefore, the allocated commands on GUI window are issued to HEW, and HEW simulation can be controlled.

The following environments can be created by Tcl/Tk.

・Issue of commands from Tcl/Tk to HEW.
  Tcl/Tk issue commands to HEW, and control simulation of HEW.

・Issue of commands from Tcl/Tk to HEW, and accepting data as the results.
  Tcl/Tk issue commands to HEW, and accepts the results and display them.

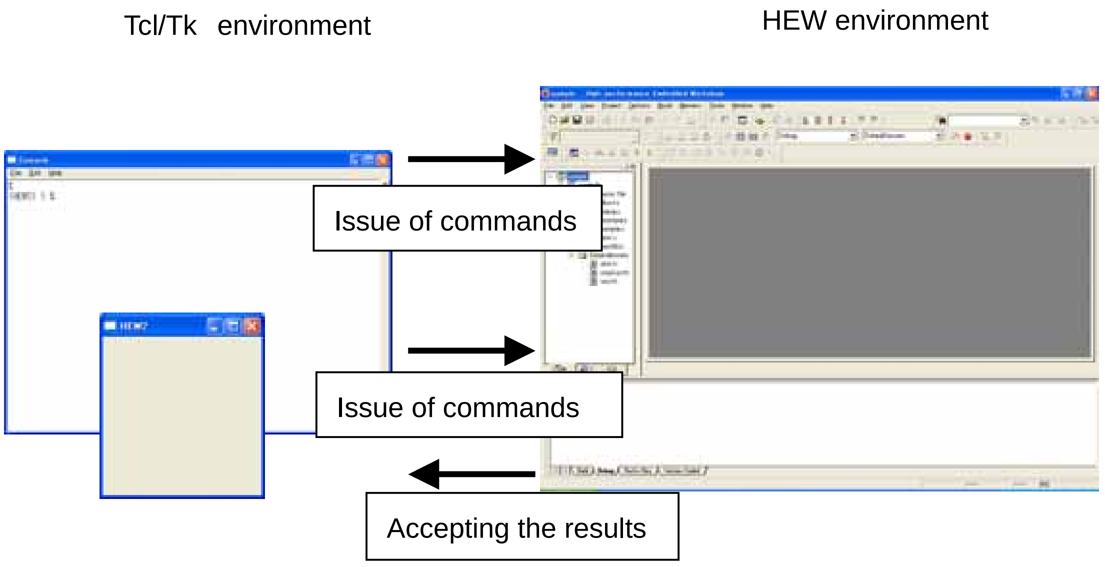
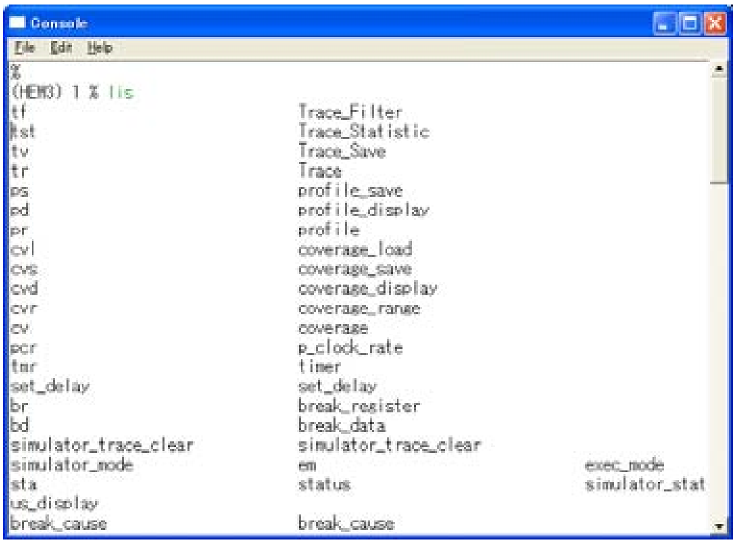


Figure: Linkage between HEW and Tcl/Tk

## 3-1, HEW commands available in Tcl/Tk

HEW offers a lot of commands, which is available in command line of HEW. You can use some of the commands to create a development environment matched to your needs.

A list of HEW commands, which are available in Tcl/Tk can be referred by inputting lis command on Console window of Tcl/Tk. Please check the meanings of the commands before using.



```
%
(HEW3) 1 % lis
tf                           Trace_Filter
tst                          Trace_Statistic
tv                           Trace_Save
tr                           Trace
ps                           profile_save
pd                           profile_display
pr                           profile
cvl                          coverage_load
cvs                          coverage_save
cvd                          coverage_display
cvr                          coverage_range
cv                           coverage
pcr                          p_clock_rate
tmr                          timer
set_delay                    set_delay
br                           break_register
bd                           break_data
simulator_trace_clear        simulator_trace_clear
simulator_mode               em                           exec_mode
sta                          status                       simulator_stat
us_display
break_cause                  break_cause
```

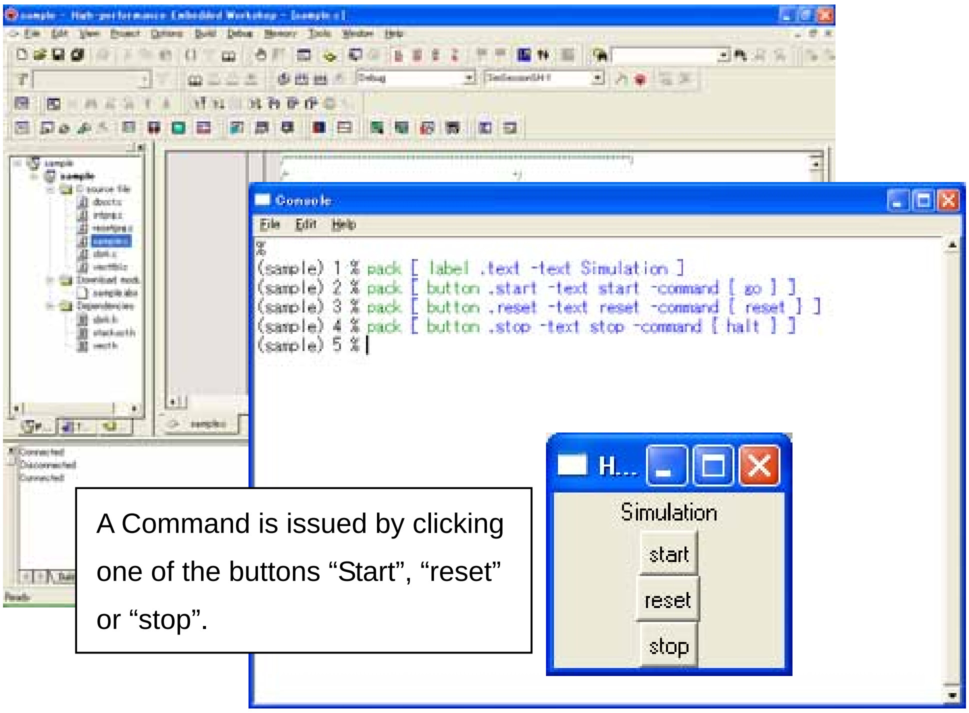
Figure: A list of HEW commands on Console window

## 3-2, Creation of Control Commands to HEW Simulator

GUI window to control HEW simulator can be created in Tcl/Tk.

```
~ sample program ~
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 }
```

In the above sample program, the simplest commands for execution of HEW simulation are allocated on the buttons on GUI window of Tcl/Tk, an environment for controlling HEW simulator is created.



```
%
(sample) 1 % pack [ label .text -text Simulation ]
(sample) 2 % pack [ button .start -text start -command [ go ] ]
(sample) 3 % pack [ button .reset -text reset -command [ reset ] ]
(sample) 4 % pack [ button .stop -text stop -command [ halt ] ]
(sample) 5 %
```

A Command is issued by clicking one of the buttons “Start”, “reset” or “stop”.

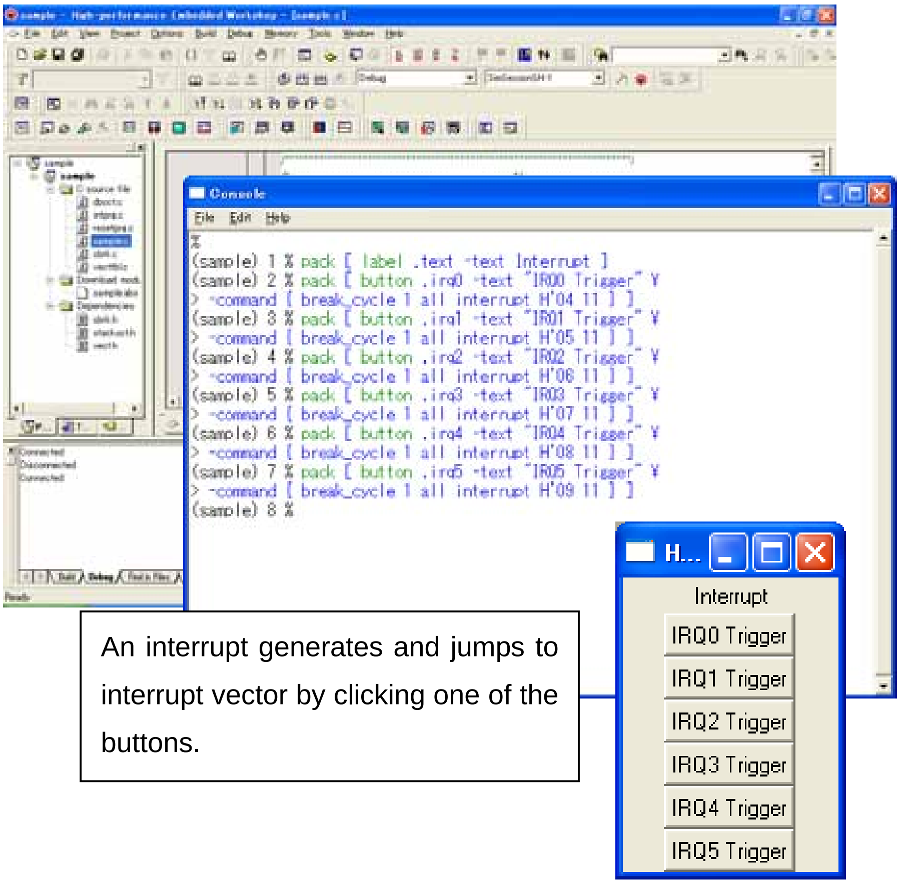
Figure: Example of execution

In the above sample program, GUI window of Tcl/Tk is created by using lavel command and button commands of Tcl/Tk. You can specify the message on the window “Simulation” using label command. You can also specify buttons to be allocated on the window , and commands to be assigned to the buttons by specifying HEW commands in the curly brackets of –command option. Therefore, one of the commands “start”, ”reset” or “halt” is issued depending on clicked button on GUI window. In the above sample program, pack command is used to allocate the buttons.

## 3-3, Creation of Command to Input Quasi Interrupt to HEW Simulator

GUI window to input interrupt signals to HEW can be created in Tcl/Tk.

```
~ sample program ~
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 }
```



```
%
(sample) 1 % pack [ label .text -text Interrupt ]
(sample) 2 % pack [ button .irq0 -text "IRQ0 Trigger" ¥
> -command { break_cycle 1 all interrupt H'04 11 } ]
(sample) 3 % pack [ button .irq1 -text "IRQ1 Trigger" ¥
> -command { break_cycle 1 all interrupt H'05 11 } ]
(sample) 4 % pack [ button .irq2 -text "IRQ2 Trigger" ¥
> -command { break_cycle 1 all interrupt H'06 11 } ]
(sample) 5 % pack [ button .irq3 -text "IRQ3 Trigger" ¥
> -command { break_cycle 1 all interrupt H'07 11 } ]
(sample) 6 % pack [ button .irq4 -text "IRQ4 Trigger" ¥
> -command { break_cycle 1 all interrupt H'08 11 } ]
(sample) 7 % pack [ button .irq5 -text "IRQ5 Trigger" ¥
> -command { break_cycle 1 all interrupt H'09 11 } ]
(sample) 8 %
```

An interrupt generates and jumps to interrupt vector by clicking one of the buttons.

Figure: Example of execution

In the sample program on the previous page, quasi interrupt function, which is prepared in HEW is used. By assigning break_cycle command, which generates a quasi interrupt to the buttons on the GUI window, it is possible generating interrupts anywhere at an execution of simulation.
BREAK commands are prepared in HEW to generate quasi interrupt.

List of BREAK commands

| Commands | Example of how commands are used |
|---|---|
| break_access | break_access <start_addr> [< end_addr>] [<mode>]<br>interrupt <interrupt_type1> <interrupt_type2> [<priority>] |
| break_cycle | break_cycle <cycle> [<count>]<br>interrupt <interrupt_type1> <interrupt_type2> [<priority>] |
| break_data | break_data <addr> <data> [<size>] [<option>]<br>interrupt <interrupt_type1> <interrupt_type2> [<priority>] |
| break_register | break_register <register> [<data> <size>] [<option>]<br>interrupt <interrupt_type1> <interrupt_type2> [<priority>] |
| break_point | break_point <addr> [<count>]<br>interrupt <interrupt_type1> <interrupt_type2> [<priority>] |

These commands must be assigned to buttons before a simulation.

In the sample program on the previous page, break_cycle 1 all …command is assigned on the buttons. (SH1 is selected as CPU ) Therefore, if a button is clicked in the middle of a simulation, a quasi interrupt generates 1 cycle later from the time when the button is clicked.

## 3-4, Creation of Environment to Control HEW simulation

One integrated environment is created by creating a script file, which is constructed by the control commands to HEW simulation.

```
~ sample program ~
#!/bin/sh
# the next line restarts using wish¥
exec tclsh "$0" "$@"

catch {destroy .top}

############################################################################
####
# CREATING WIDGETS        Window
# The false interruption command of HEW is described into the bold letter portion of each
button.
############################################################################
####
toplevel       .top
wm title       .top "HEW Simulation"
wm geometry .top 230x450+216+109; update
wm maxsize    .top 1028 753
wm minsize    .top 104 1
```

Definition of a new top level window

```
############################################################################
#####
# SETTING COMMAND        Button
# The arrangement part of each button is specified.
############################################################################
#####
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
```

Definition of buttons

```
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
##########################################################################
#####
# SETTING GEOMETRY         Button
##########################################################################
#####
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
```

Allocation of buttons

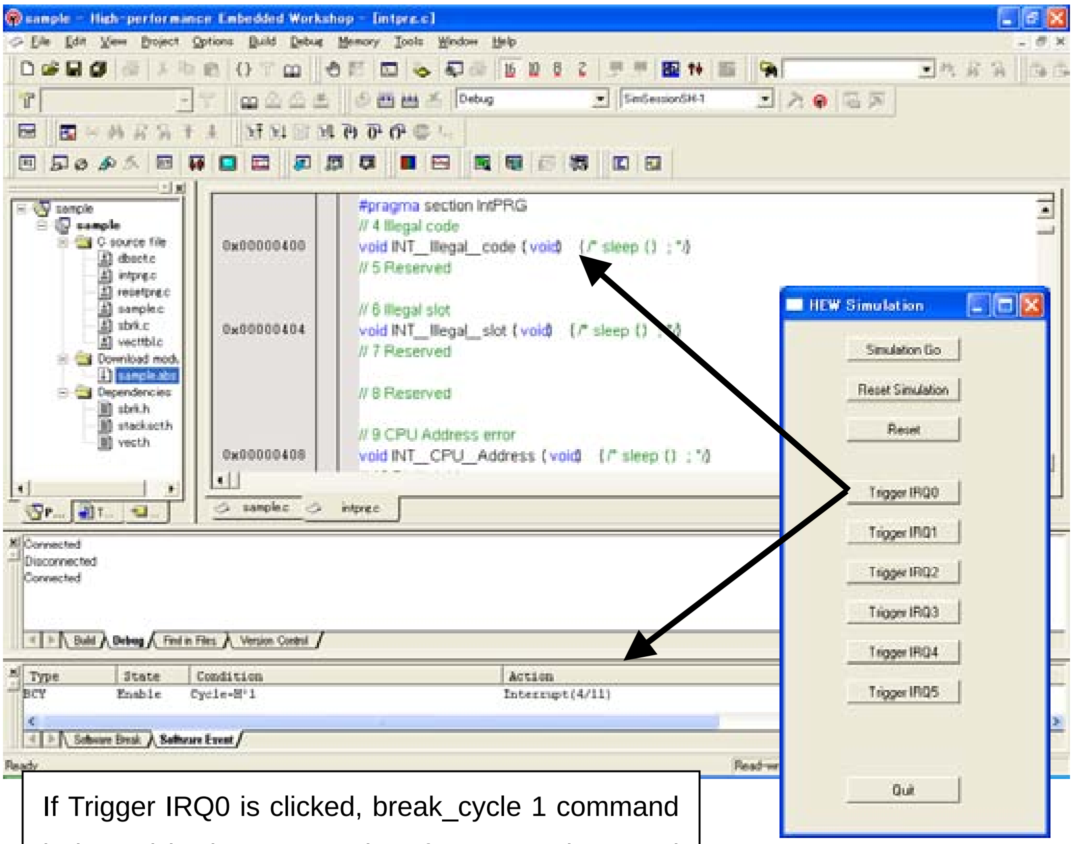
If the quasi interrupts from IRQ0 to IRQ5 are generated using the quasi interrupt function in HEW simulator, setups of the break commands remain in the simulator of HEW.    In order to execute the simulation again, the break commands must be turned off.

In the above sample program, the following commands are assigned on the buttons on GUI window in addition to the break commands.

```
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
```

These commands can be assigned by listing them in curly brackets “{}”of -command option separating each command by a semicolon “;”.  When a button is clicked, the listed commands are issued to HEW starting from the left.

In the sample program on the previous page, break_clear command is issued to HEW to turn off break_cycle command before an execution of “go” and “reset go” commands. Therefore, the simulation can be executed again.



```
0x00000400    #pragma section IntPRG
              // 4 Illegal code
              void INT_Illegal_code (void)   {/* sleep () ; */}
              // 5 Reserved

              // 6 Illegal slot
0x00000404    void INT_Illegal_slot (void)   {/* sleep () ; */}
              // 7 Reserved

              // 8 Reserved

              // 9 CPU Address error
0x00000408    void INT_CPU_Address (void)   {/* sleep () ; */}
```

```
Connected
Disconnected
Connected
```

```
Type    State    Condition    Action
BCY     Enable   Cycle=H'1    Interrupt(4/11)
```

If Trigger IRQ0 is clicked, break_cycle 1 command is issued in the event point of HEW, and a quasi interrupt is generated, then jumps to a interrupt vector.

Figure: Example of execution

## 3-5, Creation of Environment for Input Control to HEW Simulation 1

An environment for input control to HEW simulator at execution of HEW simulation.

The control environment accepts external input.

~ Sample program ~

```
#!/bin/sh

# Initial setting of an address
set addr ff800030
# Initial setting of data
set data 0
```

Set initial values for variable addr and data

```
# The command for an address setup
# An address value is stored in Variable addr.
label .l_addr –text ADDRESS
place .l_addr –x 10 –y 10

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

Create an entry box for address input

Specify the coordinates of Label and entry using place command and allocate them

```
# The command for an data setup.
# A data value is stored in Variable data
label .l_data –text DATA
place .l_data –x 10 –y 50

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

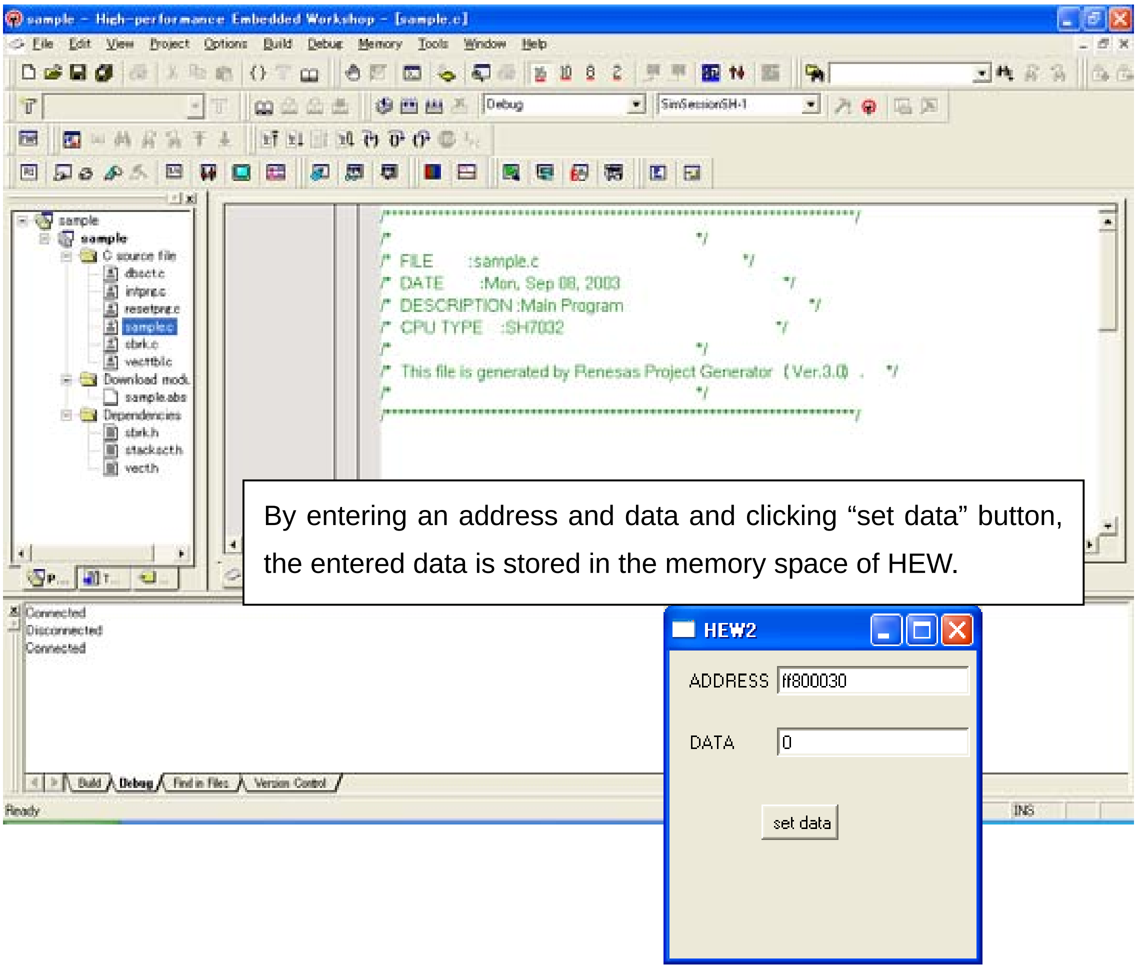
Create an entry box for data input

```
# The HEW command is set as a button.
# A push on a button sets data to arbitrary addresses.
button .set -command {memory_fill $addr $addr $data} -text {set data}
place .set –x 60 –y 100
```

Set a specified address and a data value using memory_fill command of HEW. Specify values for variable addr and data by entry command.

In the sample program on the previous page, one input control environment is created. Any address and data can be stored in HEW by using an entry box. When an address and data are entered on GUI window in hexadecimal and “set data” button is clicked, the data is stored in the specified address of the memory space of HEW.

Data can be stored using memory_fill command. Memory_fill command, which is issued to HEW can be selected in -command option of button command. Memory_fill command passes the address and the data values to HEW by referring the variable addr and data specified in entry command (”$” is used when the data is referred).



```
/***********************************************************************/
/*                                                                     */
/*  FILE        :sample.c                                              */
/*  DATE        :Mon, Sep 08, 2003                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :SH7032                                                */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.3.0).     */
/*                                                                     */
/***********************************************************************/
```

```
Connected
Disconnected
Connected
```

ADDRESS ff800030

DATA 0

By entering an address and data and clicking “set data” button, the entered data is stored in the memory space of HEW.

Figure: Example of execution

## 3-6, Creation of Environment for Input Control to HEW Simulation 2

An environment to set a value in memory space or registers of HEW can be created in Tcl/Tk.

~ sample program ~

```
wm geometry . 350x280
```

Size change of an existing window

```
# DATA
scale .scale -label DATA -from 0 -to 10000 -length 300 -variable var -orient horizontal
-tickinterval 2500 -showvalue true
place .scale -x 0 -y 0
```

Definition of a scale bar for data input

```
# ADDRESS
label .l_addr -text ADDRESS
place .l_addr -x 10 -y 100
```

Definition of an entry box for address input

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

# DATA size
label .l_size -text SIZE
place .l_size -x 10 -y 140
```

Definition of a radio button for data input

```
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
```

Specification of initial values for each variable

```
# Data modify 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
        }
}
```

Definition of a procedure for processing data, which is stored in HEW by specifying the data size.

```
# Data size 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"
        }
}
```

Definition of procedure for generating an argument (a data size) of a command issued to HEW by specifying the data size.

```
# Start address 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 ]
        }
}
```

Definition of a procedure for generating an argument (a start address) of a command issued to HEW by specifying the data size.

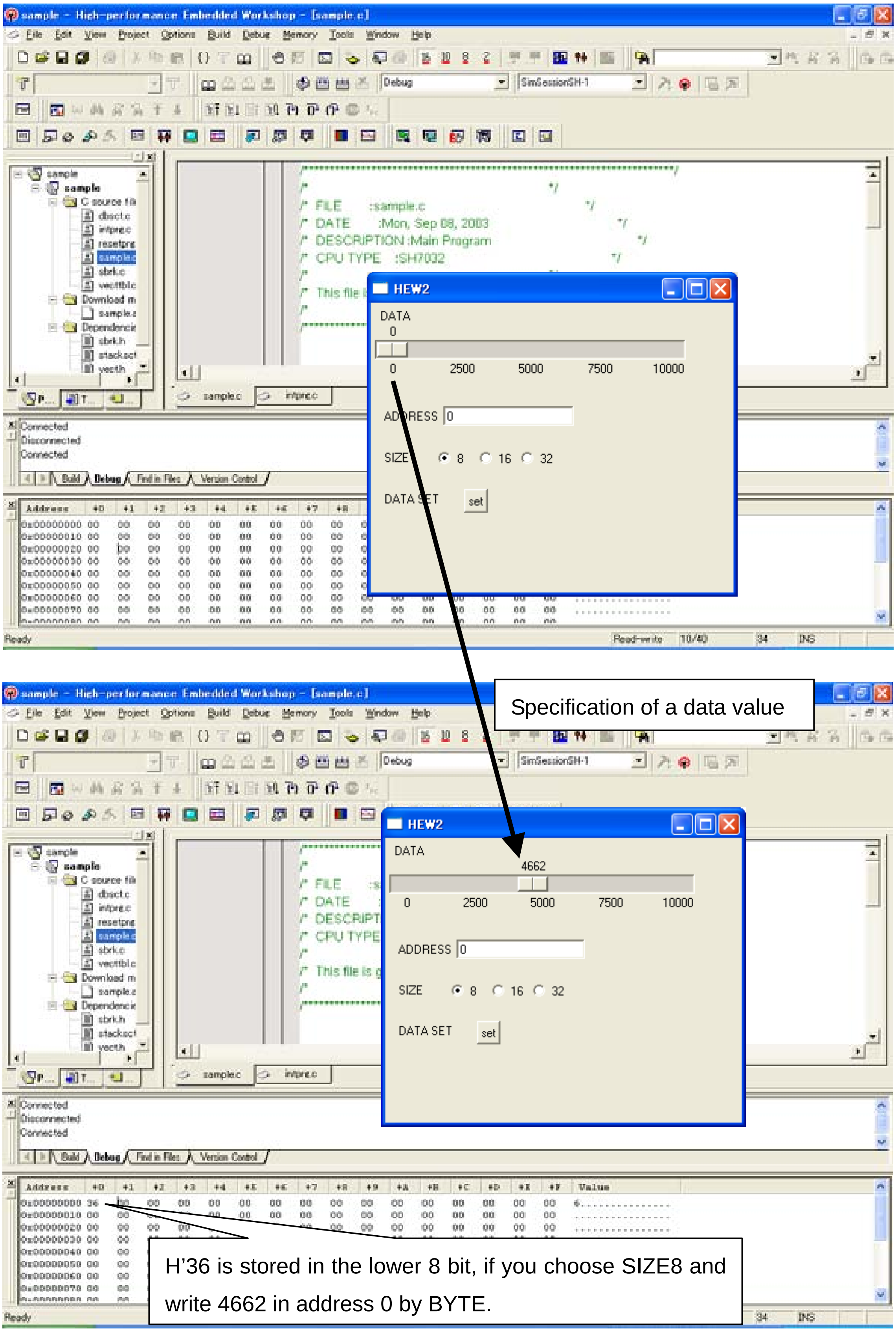
```
# End address proc
proc set_addr_e { } {
        global select
        global addr
        global addr_e
        if { $select == "size8" } {
                set addr_e [ expr $addr + 3 ]
        } elseif { $select == "size16" } {
                set addr_e [ expr $addr + 3 ]
        } else {
                set addr_e [ expr $addr + 3 ]
        }
}
```

Definition of procedure for generating an argument (an ending address) of a command issued to HEW by specifying the data size.

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

button .set -text set -command { set_size;set_addr_s;set_addr_e;set_data;memory_fill
$addr_s $addr_e $var16 $select_size }
place .set -x 90 -y 180
```

Definition of “set” button
Procedure call command and memory_fill command are assigned to the button


sample - High-performance Embedded Workshop - [sample.c]
File Edit View Project Options Build Debug Memory Tools Window Help
Debug
SimSessionSH-1
HEW2
set
sample.c
intprg.c
Build
Debug
Find in Files
Version Control
Ready

```
/*   FILE        :sample.c                  */
/*   DATE        :Mon, Sep 08, 2003         */
/*   DESCRIPTION :Main Program              */
/*   CPU TYPE    :SH7032                    */
/*   This file is
```

DATA 0

0 2500 5000 7500 10000

ADDRESS 0

SIZE ● 8 ○ 16 ○ 32

DATA SET

Connected
Disconnected
Connected

Specification of a data value

DATA 4662

0 2500 5000 7500 10000

ADDRESS 0

SIZE ● 8 ○ 16 ○ 32

DATA SET

```
Address    +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
0x00000000 36 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 6...............
0x00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

H’36 is stored in the lower 8 bit, if you choose SIZE8 and write 4662 in address 0 by BYTE.

Figure: Example of execution

The value must be specified in decimal (analog value) in the sample program. When a "set" button is clicked after entering an address value and specifying data size, the data is stored in the specified address.

H'36 is stored in the lower 8 bit, if you choose SIZE8 and write 4662 in address 0 by BYTE.
H'1236 is stored in the lower 16 bit, if you choose SIZE16 and write 4662 in address 0 by WORD.H'00001236 is stored in the 32bit, if you choose SIZE32 and write 4662 in address 0 by LONG.

The sample program is configured with setting of data, address and data size, and "set" button to set these values to HEW.

In addition, the procedures to process arguments of the data, the start address and the ending address are defined using proc command. By using these procedures in –command option of button command, arguments required for setting the data to HEW are generated at the time of clicking "set" button.

All procedures are configured simply, and they do not treat parameter passing.
The procedures can be referred by defining globally the variables used in Tcl/Tk programming.

## 3-7, Creation of Environment for Output Control from HEW simulation 1

An environment for input control to HEW simulation can be created by creating a simple script file with commands to HEW. It is because of a one-way command issue from Tcl/Tk to HEW only. However, under an environment for output control from HEW simulation, Tcl/Tk cannot accept data directly as the result of an issued command to HEW. The result must be passed to Tcl/Tk through variable.

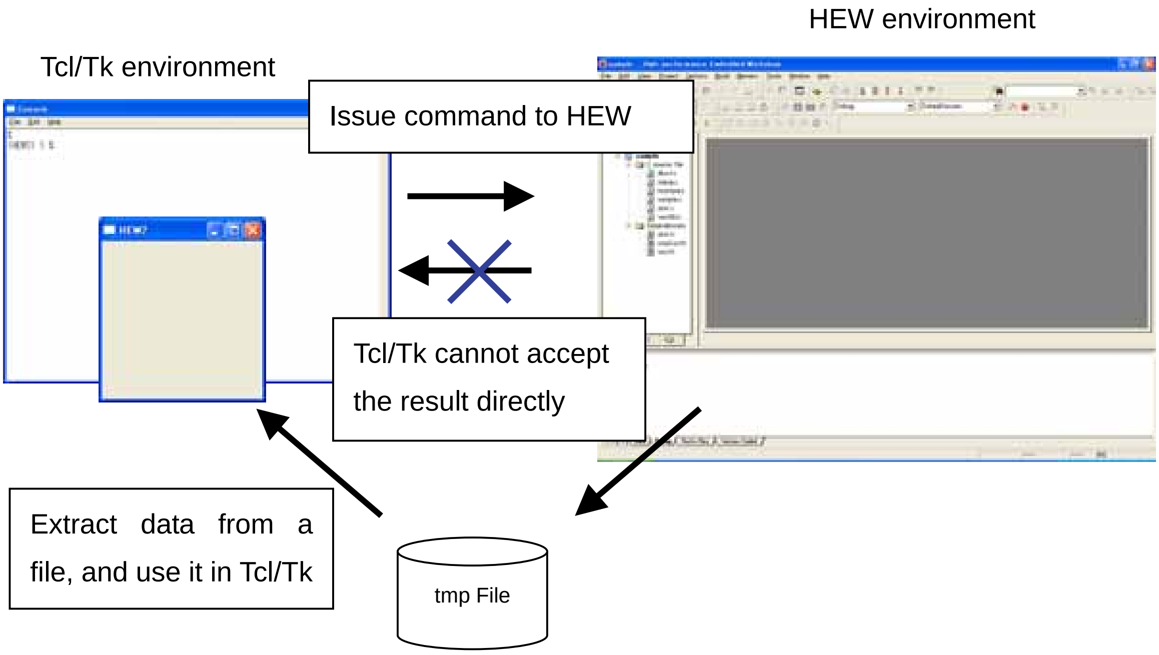

HEW environment
Tcl/Tk environment
Issue command to HEW
Tcl/Tk cannot accept the result directly
Extract data from a file, and use it in Tcl/Tk
tmp File

Figure: Output control from HEW simulator

A file” tmp.file” to temporarily store the data (the result of the issued command) must be created in advance. Tcl/Tk accepts the data by referring to information stored in the file.

## 3-8, Creation of Environment for Output Control from HEW simulation 2

This chapter describes the procedures for creating an environment for output control from HEW, which accepts the result of an issued command directly from HEW simulator at an execution of a simulation.   Under this environment, Tcl/Tk can accept value of memory and register and output them.

~ Sample program ~

```
set content "Display Address "

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

File open check

```
            # For "log.txt" checking in TEMP directory
            global env
            set dir $env(TEMP)
            set dataFile [open $dir/log.txt {RDWR TRUNC CREAT}]
            close $dataFile

            # Commandline command to read address 0x00000000
            # using HEW CommandLine command
            md 0 1
            # md command to an address to refer to here is described.

            # Read value as Console text is log into "log.txt"
            # in directory pointed by env. variable TMP
            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
```

Retrieve the data from tmp file

Append command adds a string `[lindex $d $j]`(lindex command gets the jth element of d) to a string b.

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

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

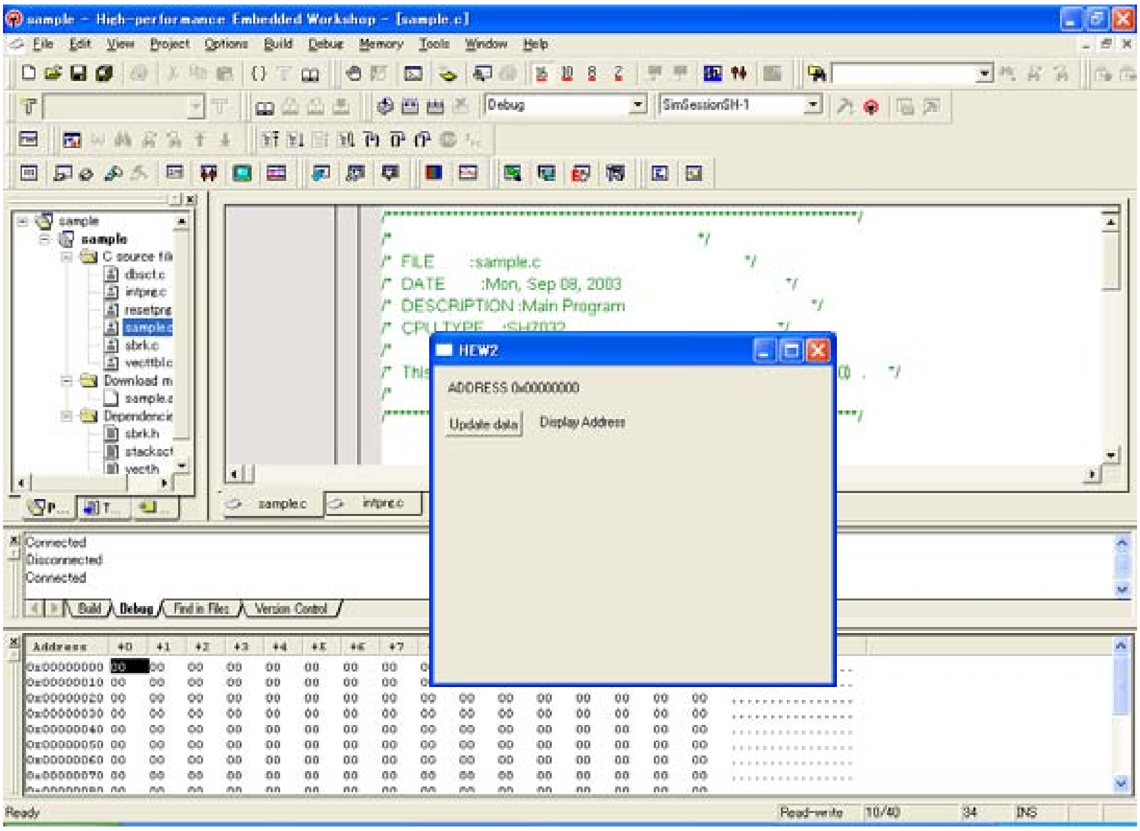
Destroy command destroys the previous result, and output the updated data.

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

place .labelcontent -x 90 -y 10

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

Allocate a button

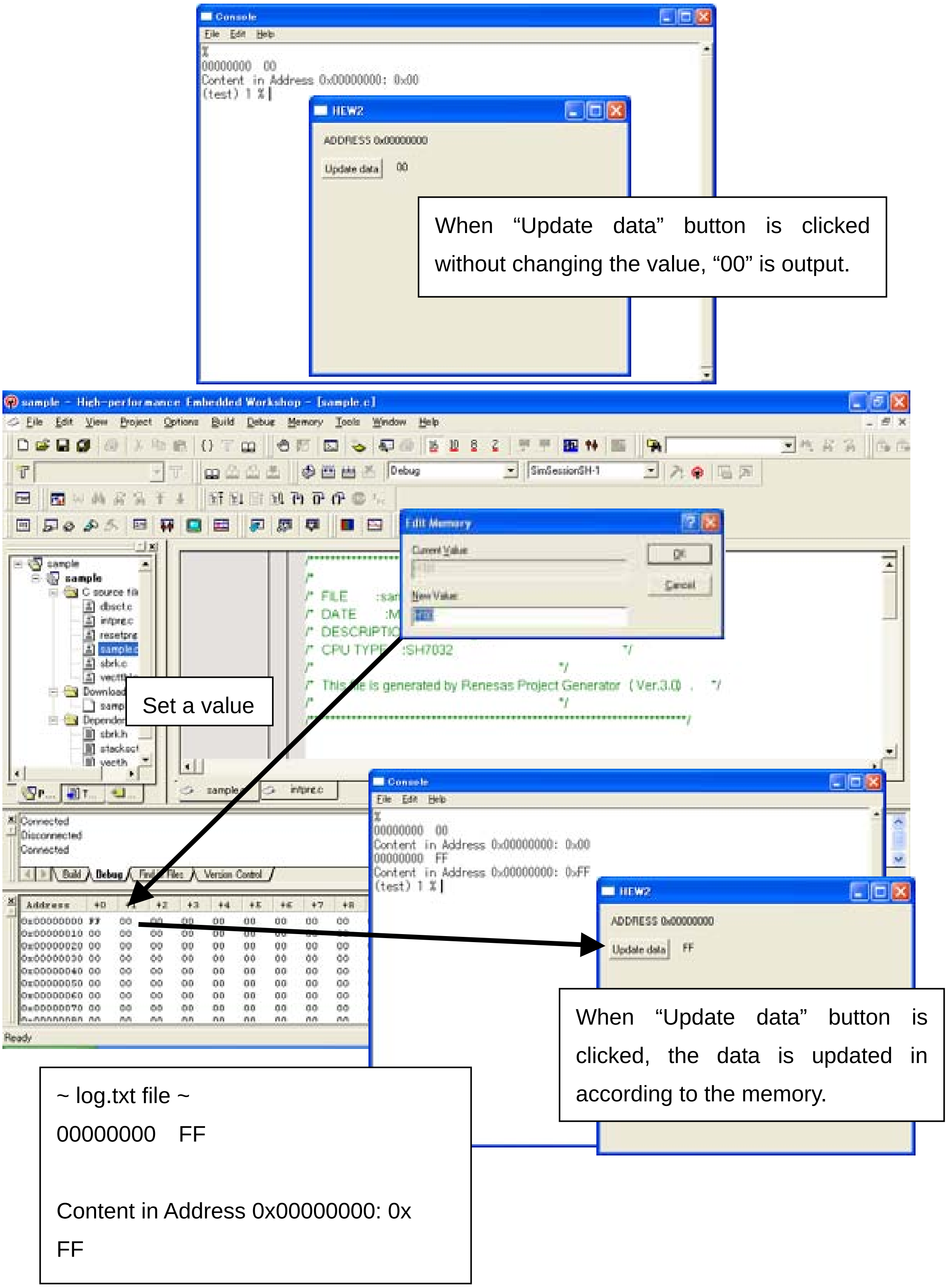


```
/*********************************************************************/
/*                                                                   */
/*  FILE        :sample.c                                            */
/*  DATE        :Mon, Sep 08, 2003                                   */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE    :SH7032                                              */
```

```
Connected
Disconnected
Connected
```

```
Address    +0  +1  +2  +3  +4  +5  +6  +7
0x00000000 00  00  00  00  00  00  00  00
0x00000010 00  00  00  00  00  00  00  00
0x00000020 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
0x00000030 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
0x00000040 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
0x00000050 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
0x00000060 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
0x00000070 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ................
```

Figure: Start-up display



```
%
00000000  00
Content in Address 0x00000000: 0x00
(test) 1 %
```

When "Update data" button is clicked without changing the value, "00" is output.

```
/* FILE        :sa
/* DATE        :M
/* DESCRIPTIO
/* CPU TYPE    :SH7032                                  */
/*                                                      */
/* This file is generated by Renesas Project Generator (Ver.3.0).   */
/*                                                      */
```

```
Connected
Disconnected
Connected
```

```
Address    +0 +1 +2 +3 +4 +5 +6 +7 +8
0x00000000 FF 00 00 00 00 00 00 00 00
0x00000010 00 00 00 00 00 00 00 00 00
0x00000020 00 00 00 00 00 00 00 00 00
0x00000030 00 00 00 00 00 00 00 00 00
0x00000040 00 00 00 00 00 00 00 00 00
0x00000050 00 00 00 00 00 00 00 00 00
0x00000060 00 00 00 00 00 00 00 00 00
0x00000070 00 00 00 00 00 00 00 00 00
```

Set a value

```
%
00000000  00
Content in Address 0x00000000: 0x00
00000000  FF
Content in Address 0x00000000: 0xFF
(test) 1 %
```

When "Update data" button is clicked, the data is updated in according to the memory.

~ log.txt file ~

00000000    FF

Content in Address 0x00000000: 0x
FF

Figure: Example of execution

HEW Tcl/Tk Application Note

Publication Date: Nov. 5, 2003 Rev.1.00

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.

Edited by: Microcomputer Tool Development Department
Renesas Solutions Corp.

# HEW Tcl/Tk
# Application Note

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan