お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

Application Note

RENESAS

HEW Tcl/Tk

アプリケーションノート

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2003.10

本資料ご利用に際しての留意事項

１．本資料は、お客様に用途に応じた適切な弊社製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について弊社または第三者の知的財産権その他の権利の実施、使用を許諾または保証するものではありません。

２．本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例など全ての情報の使用に起因する損害、第三者の知的財産権その他の権利に対する侵害に関し、弊社は責任を負いません。

３．本資料に記載の製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。

４．本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例などの全ての情報は本資料発行時点のものであり、弊社は本資料に記載した製品または仕様等を予告なしに変更することがあります。弊社の半導体製品のご購入およびご使用に当たりましては、事前に弊社営業窓口で最新の情報をご確認頂きますとともに、弊社ホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意下さい。

５．本資料に記載した情報は、正確を期すため慎重に制作したものですが、万一本資料の記述の誤りに起因する損害がお客様に生じた場合においても、弊社はその責任を負いません。

６．本資料に記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を流用する場合は、流用する情報を単独で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断して下さい。弊社は、適用可否に対する責任は負いません。

７．本資料に記載された製品は、各種安全装置や運輸・交通用、医療用、燃焼制御用、航空宇宙用、原子力、海底中継用の機器・システムなど、その故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのあるような機器・システムや特に高度な品質・信頼性が要求される機器・システムでの使用を意図して設計、製造されたものではありません（弊社が自動車用と指定する製品を自動車に使用する場合を除きます）。これらの用途に利用されることをご検討の際には、必ず事前に弊社営業窓口へご照会下さい。なお、上記用途に使用されたことにより発生した損害等について弊社はその責任を負いかねますのでご了承願います。

８．第７項にかかわらず、本資料に記載された製品は、下記の用途には使用しないで下さい。これらの用途に使用されたことにより発生した損害等につきましては、弊社は一切の責任を負いません。
　１）生命維持装置。
　２）人体に埋め込み使用するもの。
　３）治療行為（患部切り出し、薬剤投与等）を行なうもの。
　４）その他、直接人命に影響を与えるもの。

９．本資料に記載された製品のご使用につき、特に最大定格、動作電源電圧範囲、放熱特性、実装条件およびその他諸条件につきましては、弊社保証範囲内でご使用ください。弊社保証値を越えて製品をご使用された場合の故障および事故につきましては、弊社はその責任を負いません。

１０．弊社は製品の品質および信頼性の向上に努めておりますが、特に半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。弊社製品の故障または誤動作が生じた場合も人身事故、火災事故、社会的損害などを生じさせないよう、お客様の責任において冗長設計、延焼対策設計、誤動作防止設計などの安全設計（含むハードウエアおよびソフトウエア）およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特にマイコンソフトウエアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願い致します。

１１．本資料に記載の製品は、これを搭載した製品から剥がれた場合、幼児が口に入れて誤飲する等の事故の危険性があります。お客様の製品への実装後に容易に本製品が剥がれることがなきよう、お客様の責任において十分な安全設計をお願いします。お客様の製品から剥がれた場合の事故につきましては、弊社はその責任を負いません。

１２．本資料の全部または一部を弊社の文書による事前の承諾なしに転載または複製することを固くお断り致します。

１３．本資料に関する詳細についてのお問い合わせ、その他お気付きの点等がございましたら弊社営業窓口までご照会下さい。

HEW Tcl/Tk
アプリケーションノート

Rev. 1.0

目次

HEW 機能における Tcl/Tk の概要

SH、H8 のプログラム開発をサポートする開発統合環境 HEW（High-performance Embedded Workshop）では、スクリプト言語である Tcl/Tk をサポートしております。
サポートしている Tcl/Tk のバージョンは、Tcl/Tk Version 8.4.1 です。

Tcl/Tk は、Tool Command Language であるスクリプト言語のクラスメンバ“ Tcl ”とグラフィカルユーザインタフェースをプログラミングするためのツールキット“ Tk”で構成されております。スクリプト言語である Tcl/Tk は、コンパイルを不要とし作成したプログラムがすぐに反映されていきます。

Tcl は、単純なプログラミングを可能にする文法を持っており、スタンド・アロンのアプリケーションとして使用したり、アプリケーションプログラムに組み込んで使用したりする事ができます。
Tk は、ユーザ独自の GUI を迅速に構築することを可能にします。

HEW では Tcl/Tk をサポートする事により、Tcl/Tk のプログラミングにより HEW で本来提供しています機能や GUI 以外に、ユーザの御使用方法に合わせた開発環境として GUI 環境のカスタマイズを行う事を可能にしています。

1、Tcl/Tk の使用方法

1-1、Tcl/Tk の起動

HEW 上で Tcl/Tk コマンドを御使用いただく方法を御説明します。
HEW 起動後に HEW のコマンドメニューより“表示”－“TCL ツールキット”を選択します。Tcl/Tk のプログラミングをサポートする Console ウィンドウと GUI ウィンドウが起動します。ユーザは、これらのウィンドウを用いて Tcl/Tk のプログラミングと実行を行っていく事が可能です。

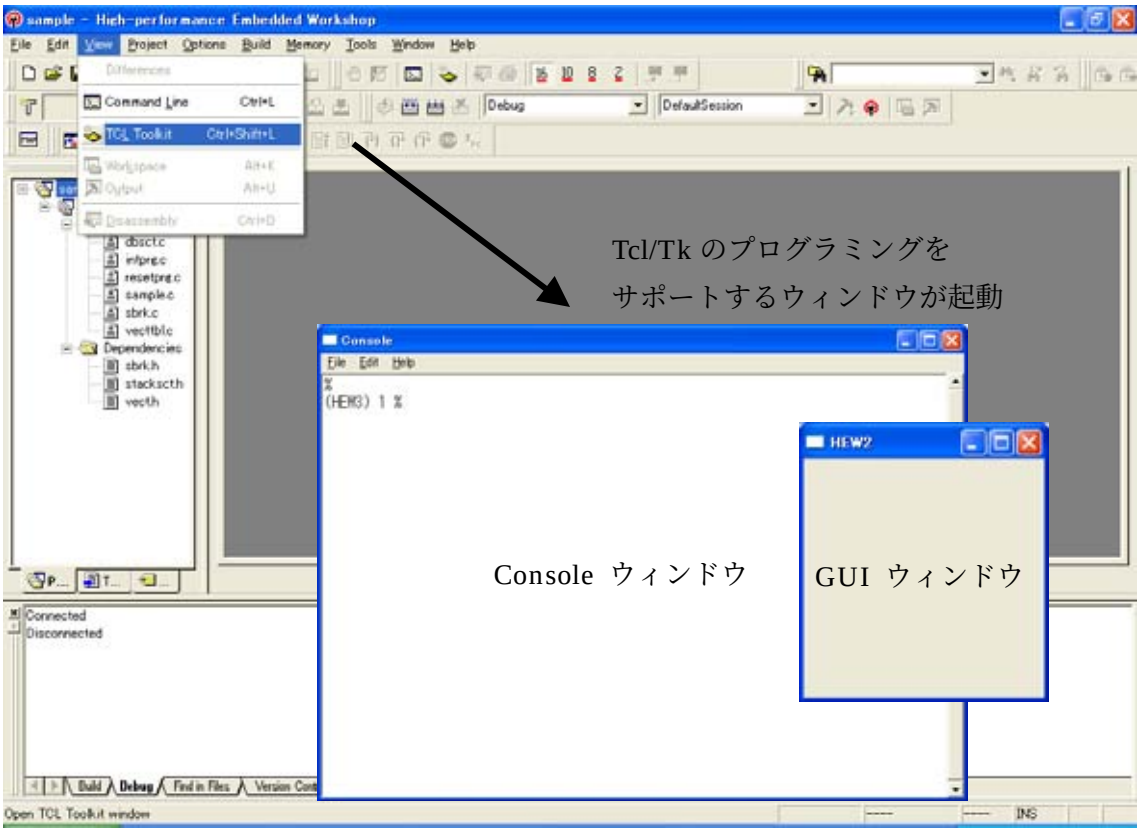


図 Tcl/Tk の起動

Console ウィンドウ
Console ウィンドウで、Tcl/Tk のプログラミングと実行を行っていきます。ユーザは、インタプリタ形式でプログラミングと実行を行っていく事が可能です。また本ウィンドウにて予め作成しておいたスクリプトファイルをロードして実行する事も可能です。

GUI ウィンドウ
Console ウィンドウで実行した結果が反映されます。HEW では、デフォルトで Toplevel

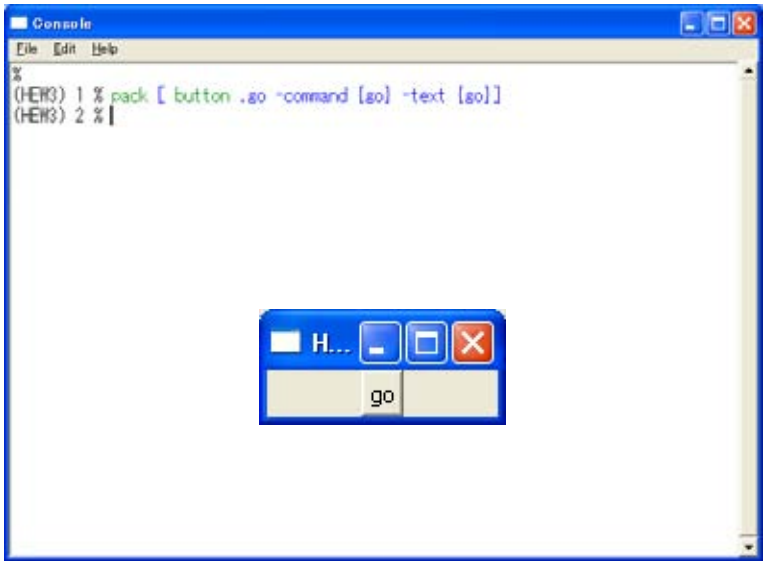
ウィンドウをご用意しています。また、本ウィンドウ以外に新規に GUI ウィンドウを作成していく事も可能です。

1-2、Tcl/Tk の実行方法

HEW 上で Tcl/Tk の実行を行うには、次のような方法があります。

① インタプリタ形式での実行

インタプリタ形式でのプログラミングでは、Console ウィンドウに Tcl/Tk のコマンドを入力していただきます。Tk のコマンドを入力した場合には、入力した内容に応じて Tcl/Tk ツールキット起動時の GUI ウィンドウが反映されます。



例）HEW のプログラム実行コマンド“ go ”をボタンに割り付けた例

図インタプリタ形式でのプログラミング

② スクリプトファイルをロードすることによる実行

予め Tcl/Tk のプログラムをスクリプトファイルで作成し、Console ウィンドウでファイルをロードする事が可能です。Console ウィンドウのコマンドメニューで“ File ”―“ Source ”を選択します。“ Select a file to source ”ウィンドウが起動しますので作成しておいた Tcl/Tk のスクリプトファイルを選択します。

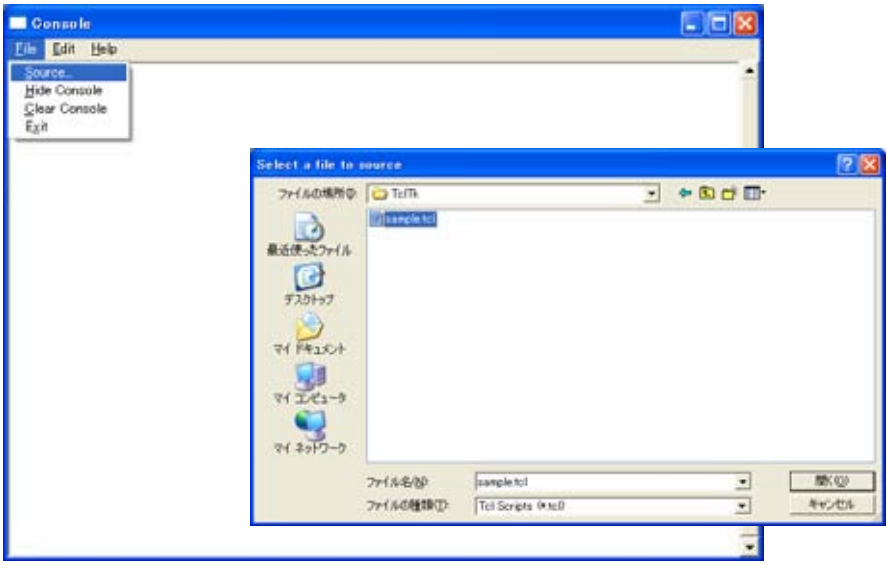
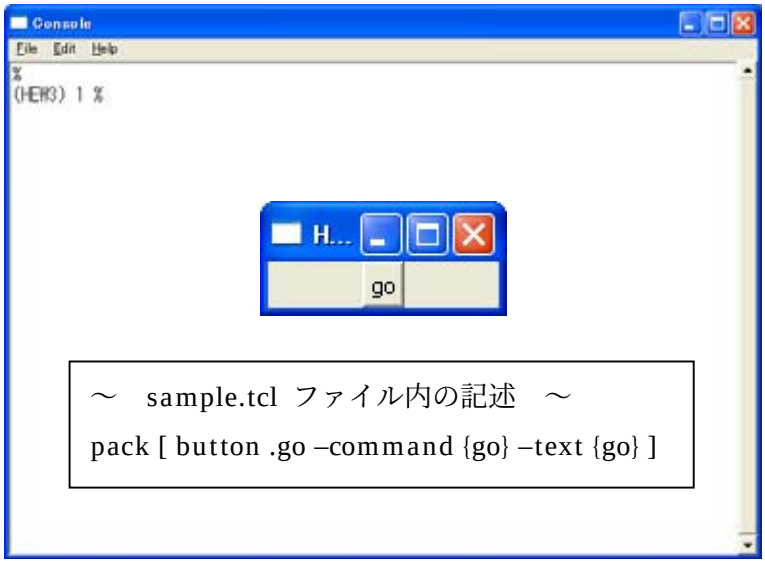


図 Tcl/Tk　ソースファイルの選択方法

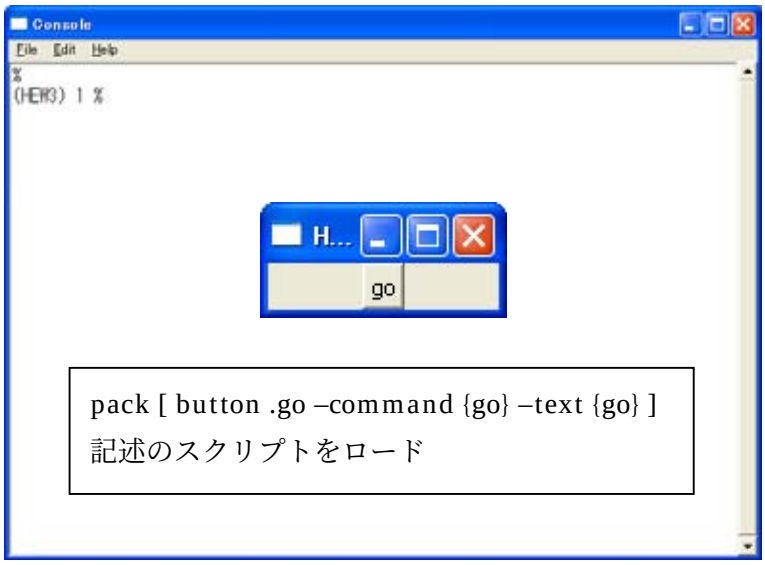


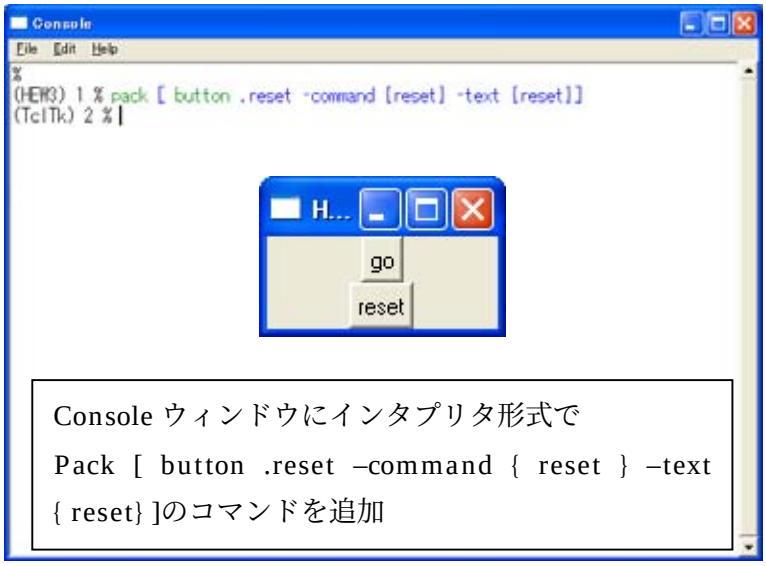
図ソースファイル選択後

ファイル選択後には、スクリプトファイル内のプログラム内容が反映されます。

③ インタプリタ形式とスクリプトロード形式の併用による実行

予めスクリプトファイルで作成しておいたファイルをロードした後にインタプリタ形式でプログラムの追加を行っていく事が可能です。





例）HEW のリセットコマンド“ reset ”を追加

図　プログラム追加

2、Tcl/Tk の基本的なプログラミング方法

2-1 Tcl のプログラミング方法

2-1-1、基本的な文法例

Tcl は、非常に簡単な文法で成り立っています。Tcl のビルドインコマンドやユーザが作成したプロシージャを Command として使用してプログラミングを行っていきます。プログラムの記述方法は、Command と Command に必要な引数をスペースで区切りひとつの実行形態を構成します。また、改行やセミコロンによって区切る事で複数の実行形態を記述していきます。

基本的な文法 1：引数はスペースで区切って並べます。

```
Command arg1 arg2 arg3 …
```

基本的な文法 2： 一行に複数のコマンドを並べる場合はセミコロン(;)で区切ります。

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

基本的な文法３： 複数行にひとつのコマンドを並べる場合は¥記号を用います。

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

基本的な文法４： コメントを追記する場合には（#）を用います。

```
Command arg1            #コメント
```

Command は、Tcl のビルドインコマンドやユーザ作成のプロシージャ
arg は、ビルドインコマンドやプロシージャに必要となる引数です。

2-2、Tcl のビルドインコマンド

Tcl に準備されているビルドインコマンドをご説明します。一般的に使用される最低限のビルドインコマンドをご紹介します。

2-2-1、変数

Tcl で使用する変数は、使用する前に型を宣言する必要がありません。変数としてユーザが任意の名前を定義して使用する事が可能です。Tcl のプログラム内では、変数へ値をセットする事や変数の値を参照する事が可能です。

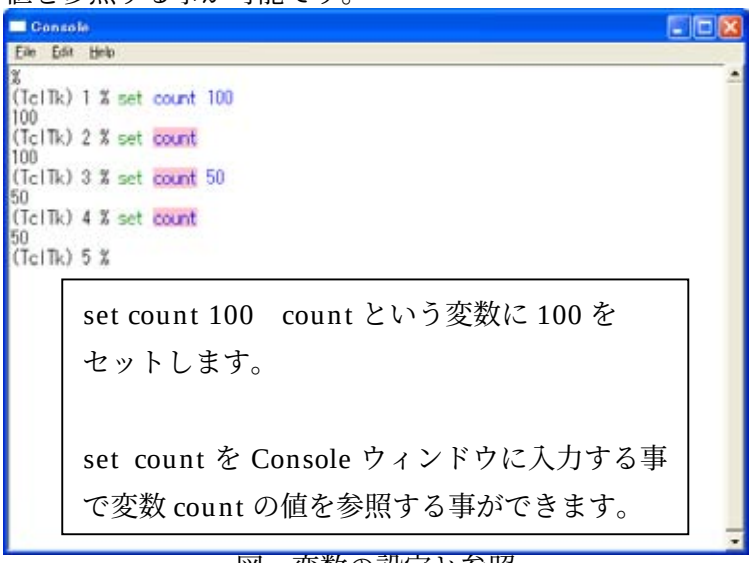


図　変数の設定と参照

また Tcl では、変数に任意の文字列を設定する事ができます。

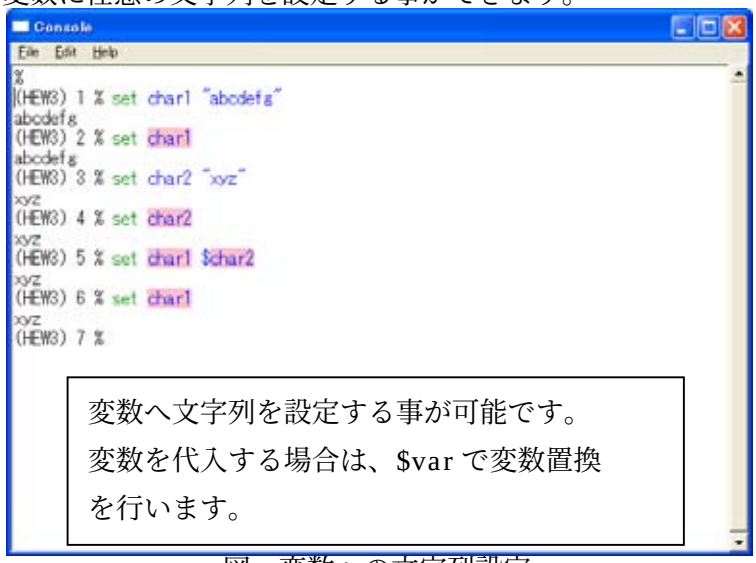


図　変数への文字列設定

2-2-2、配列

Tcl では、配列を使用する事が可能です。配列への代入は、配列の個別要素への代入や配列全体への一括代入をすることが可能です。

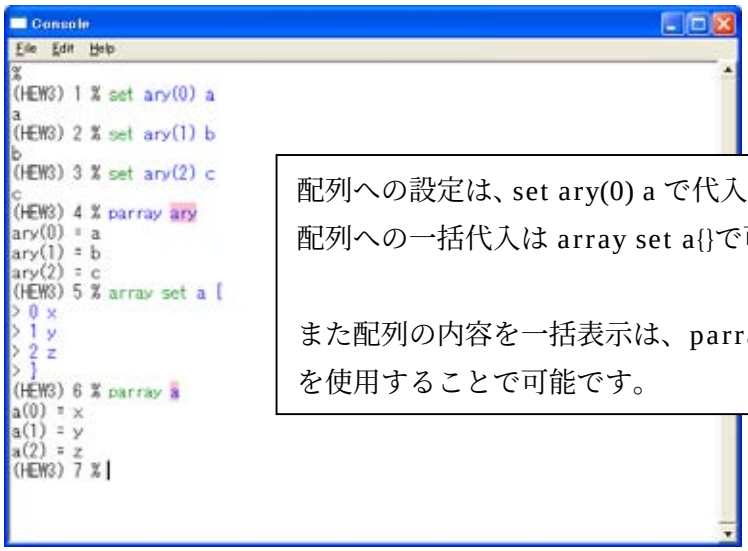


図　配列

2-2-3、算術演算

Tcl では、expr コマンドを使用する事で整数や浮動小数点の演算や大小比較等を実行する事が可能です。Tcl では、ビルドインコマンドとして演算子や数学関数が準備されています。

図　算術演算

Tcl でサポートしている演算子や関数を表に示します。

表　演算子

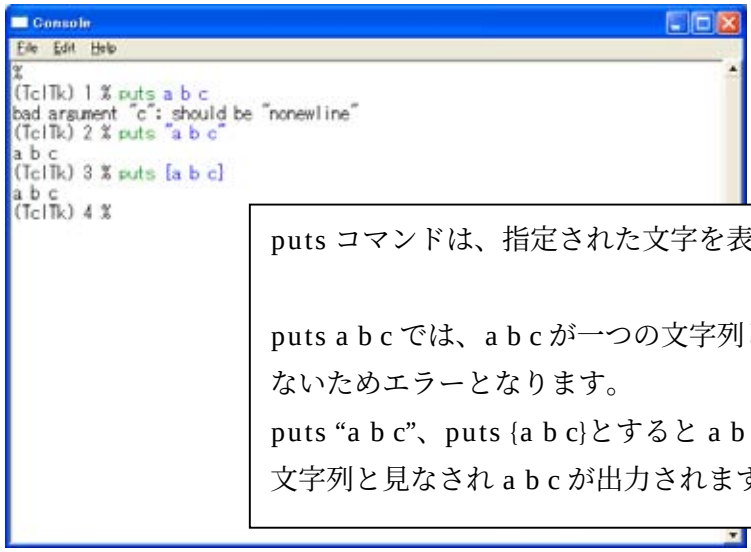
記号	記号内容
-、+、 ^ 、!	不符号、正符号、補数、否定
*、/、%	乗算、除算、剰余
+、-	加算、減算
<<、>>	左シフト、右シフト
<、>、<=、>=	ブール式の比較（左不等、右不等、等価左不等、等価右不等）
==、!=	ブール式の等号、不等号
eq、ne	ブール式の等号、不等号（文字列で使用）
&、^	ビット積（AND)、ビット差（XOR）
&&、||	積結合、和結合
x?y:z	条件式

表　関数

関数	関数内容
acos、cos、hypot、sinh、asin()、cosh()、log()、sqrt()、atan()、exp()、log10()、tan()、atan2()、floor()、pow()、tanh()、ceil()、fmod()、sin()、	数学関数
abs(arg)	絶対値
double(arg)	倍制度値
int(arg)	整数値
Rand()	乱数値
round(arg)	整数への丸め値
srand(arg)	乱数の初期値

2-2-4、ダブルクォートと中括弧

Tcl では、ダブルクォート“ ”” ”と中括弧“ ( ) ”で括られた文字列をひとつの文字列として扱います。



puts コマンドは、指定された文字を表示します。

puts a b c では、a b c が一つの文字列と見なされないためエラーとなります。
puts “a b c”、puts {a b c}とすると a b c が一つの文字列と見なされ a b c が出力されます。

図　ダブルクォートと中括弧

2-2-5、コマンドの代入

Tcl では、大括弧“ [　] ”で括られた文字列を一つのコマンドとして扱います。

expr 100/20 を大括弧“ [　] ”で括ると Tcl は expr 100/20 を一つのコマンドとして扱います。この結果、set コマンドで計算結果である 5 が表示されます。

また set コマンドを使用している為、expr 100/20 を一つのコマンド command に設定します。
この為、set command と実行すると同じ結果が得られます。

また expr 100/20 を“ [ ] ”で括らない場合はエラーとなります。

図　コマンドの代入

2-2-6、入出力フォーマット

Tcl は、入出力コマンドとして scan、format コマンドを準備しています。本コマンドは、ANSI の scanf、printf に対応したコマンドとなっています。また、%によって書式のフォーマットを規定する事ができます。

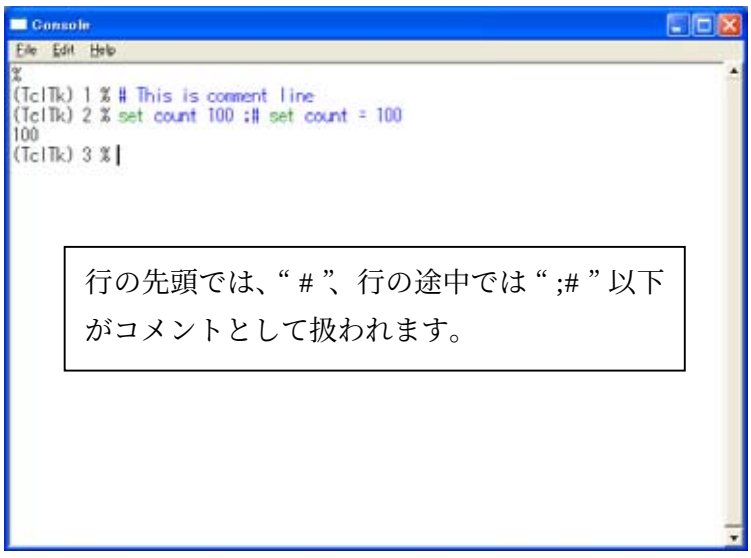
scan 3.1415926 “%d.%d” int float は、ANSI の scanf に相当します。本コマンドは、3.1415926 の整数部 3 と少数部 1415926 をそれぞれ int、float 変数に取り込みます。また、この実行結果として入力した数値の個数 2 を返します。

format “%d.%d” $int $float は、ANSI の printf に相当します。本コマンドの実行結果は、3.1415926 を返します。

図　入出力フォーマット

2-2-7、コメントの記述

Tcl のスクリプトファイルにコメントを記述する場合は、行の先頭に “ # ” を記述します。また、行の途中からコメントを記述する場合は “ ;# ” となります。



行の先頭では、“ # ”、行の途中では “ ;# ” 以下がコメントとして扱われます。

図　コメントの記述

2-2-8、プロシージャ

Tcl では、ユーザが任意のコマンドの羅列を C 言語の関数のようにして扱う事が可能です。Tcl でプロシージャと呼ばれる関数は、Tcl のビルドインコマンドと同等に使用する事ができます。

proc コマンドを使用し 0 個以上の引数を取る関数としてプロシージャを作成する事が可能です。プロシージャ内で定義された変数は、ローカル変数として扱われプロシージャ内のみで参照する事が可能です。またプロシージャからグローバル変数を参照する場合は、プロシージャ内にグローバル変数の定義が必要になります。

プロシージャ作成例

```
～　サンプルプログラム　～
set count3 100                ;#外部変数として変数 count3 に 100 を代入します。
proc add{ count1 count2 }{    ;#引数　count1、count2 を受け取ります。
        global count3         ;#count3 を参照する為、global 定義が必要です。
        return[ expr $count1+$count2+$count3]
                              #プロシージャ add が返す値を return で定義します。
}
add 200 300                   ;#プロシージャ　add を使用します。
```

本プログラムを入力するとプロシージャが呼ばれる add 200 300 の実行結果として 600 が得られます。

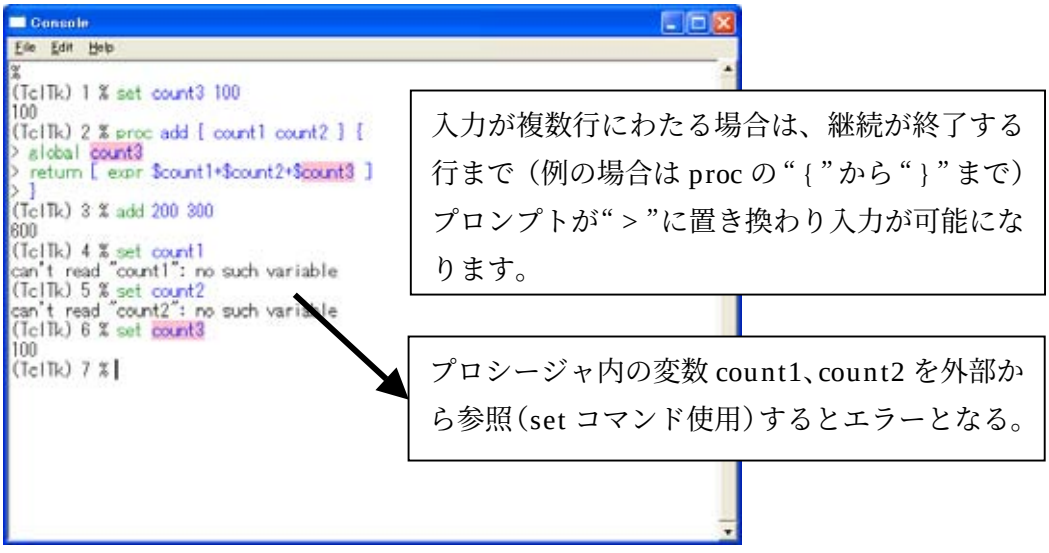


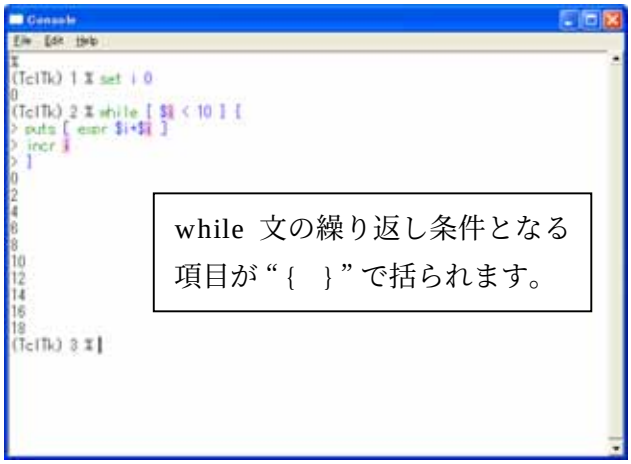
図　プロシージャ作成

2-2-9、制御構造コマンド

Tcl は、他の高級言語に存在する重要な制御構造をサポートしています。
while、for、if、if...else、switch、foreach 等を使用する事ができます。

① while 使用例

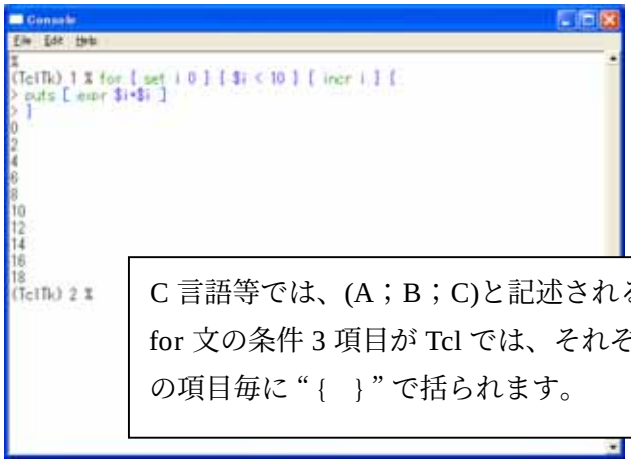
```
set i 0
while { $i < 10 } {
        puts [ expr $i+$i ]
        incr i
}
```



while 文の繰り返し条件となる
項目が“{ }”で括られます。

② for 使用例

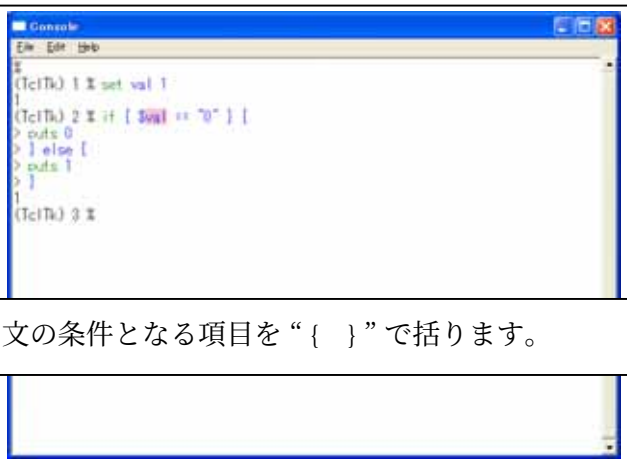
```
for { set i 0 } { $i < 10 } { incr i } {
        puts [ expr $i+$i ]
}
```



C 言語等では、(A；B；C)と記述される
for 文の条件 3 項目が Tcl では、それぞれ
の項目毎に“{ }”で括られます。

③ if…else 使用例

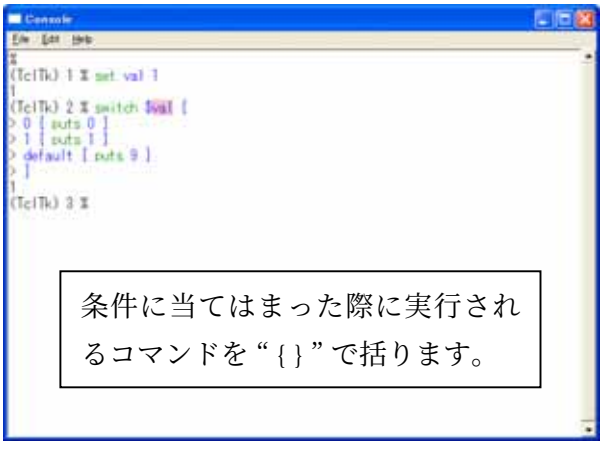
```
set val 1
if { $val == “0” } {
        puts 0
} else {
        puts 1
}
```



if 文の条件となる項目を “{　}” で括ります。

④ switch 使用例

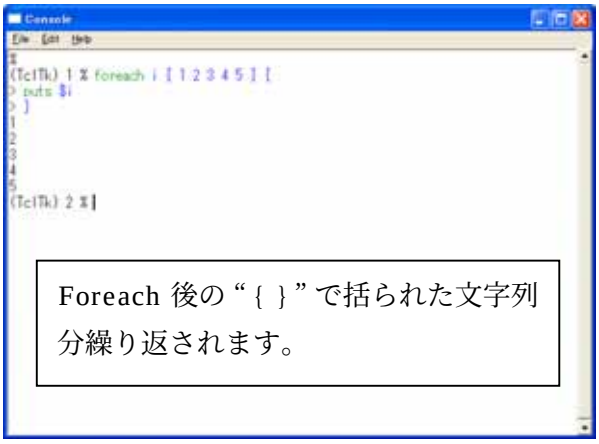
```
set val 1
switch $val {
        0 { puts 0 }
        1 { puts 1 }
        default { put 9 }
}
```



条件に当てはまった際に実行されるコマンドを “{}” で括ります。

⑤ foreach 使用例

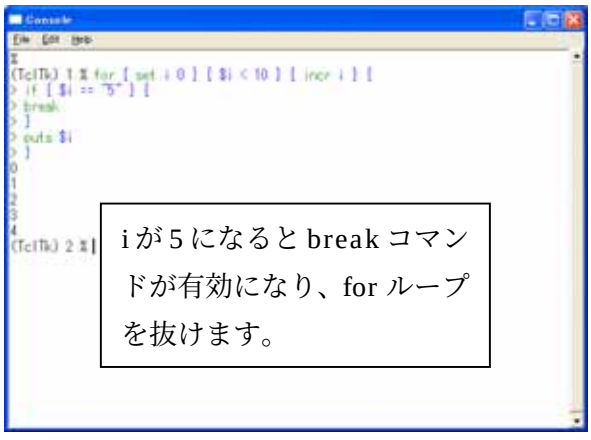
```
foreach i { 1 2 3 4 5 } {
        puts $i
}
```



Foreach 後の“{ }”で括られた文字列分繰り返されます。

⑥ while、for、foreach 内では、continue、break を使用することも可能です。

```
for { set i 0 } { $i < 10 } { incr i } {
        if { $i == “5” }{
                break           ;#break の記述が可能
        }
        puts $i
}
```



i が 5 になると break コマンドが有効になり、for ループを抜けます。

2-3、Tk のプログラミング方法

2-3-1、基本的な文法例

Tk のプログラミングでは、widget（ウィジット：Tk がサポートしている GUI の部品）を定義し作成する GUI 環境に配置していきます。

Tk の文法で使用する widget の定義は、widget とパス、オプションをスペースで区切って記述していきます。

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

パスは“ . ”で始まる任意の名前を指定します。widget コマンドが正しく実行されると.path 名のコマンドが生成されますので、このコマンドを作成したウィンドウに配置し GUI 環境を構築していきます。

```
pack .path
```

widget を配置するコマンドである pack コマンド等を使用して GUI 上に配置

Tk を用いたプログラミングでは、提供されている widget を使用してユーザ独自の様々な GUI 環境の構築が可能です。

2-4、Tk の widget

2-4-1 ボタンの作成

Tk では、button コマンドを使用する事でボタンを作成する事ができます。

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

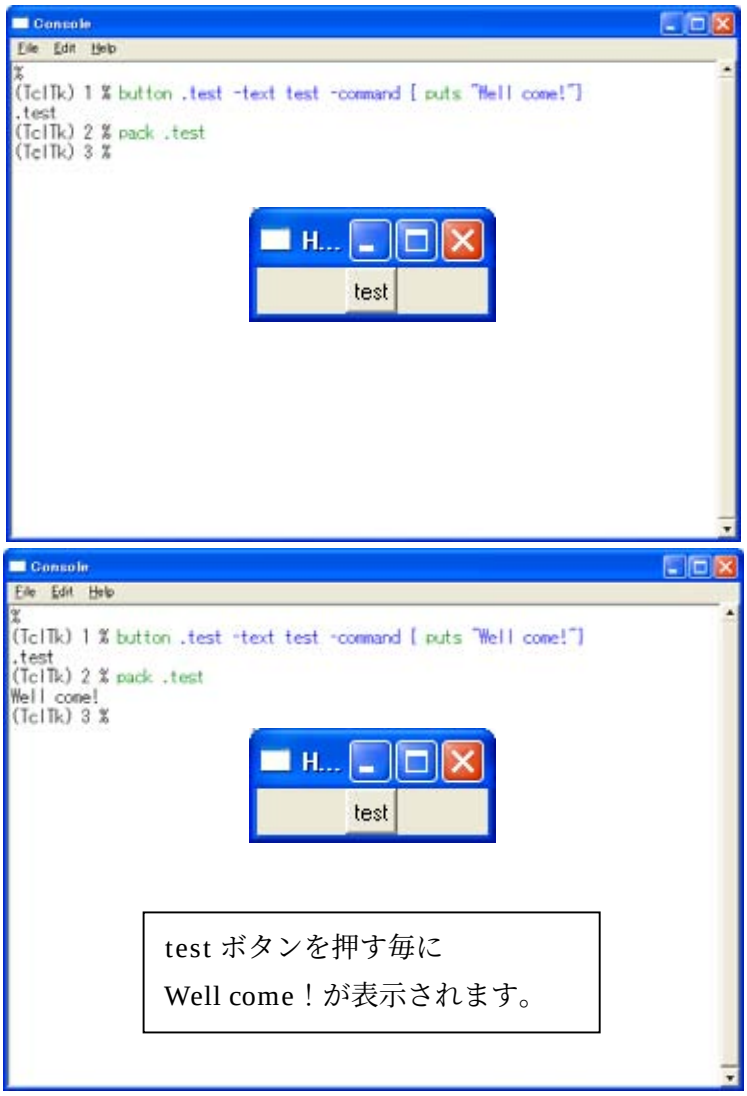


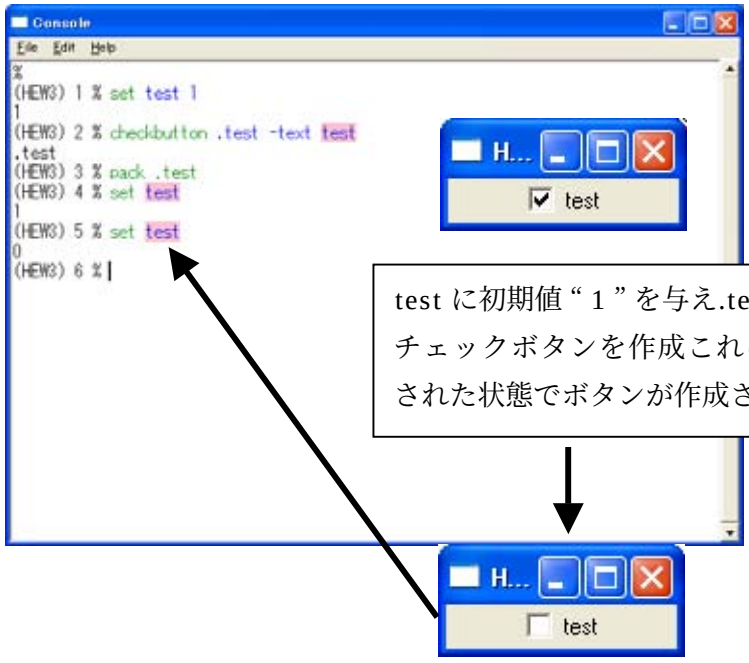
図　ボタンの作成

サンプルでは、-text オプションで指定した”test”という名前のボタンを作成しています。作成したボタンは、-command オプションにボタンが押された時に実行されるコマンドを設定しています。これにより test ボタンが押されると“{}”内のコマンドが実行され Console ウィンドウに Well come!が出力されます。

2-4-2、チェックボタンの作成

Tk では、checkbutton コマンドを使用する事でチェックボタンを作成する事が可能です。

```
set test 1                              ;#変数 test に初期値 “ 1 ” を与えます。
checkbutton .test –text test
pack .test
```



test に初期値 “ 1 ” を与え.test というパス名のチェックボタンを作成これによりチェックがされた状態でボタンが作成されます。

チェックをはずして、set コマンドで test の値を確認すると”0”になっています。

図　チェックボタンの作成

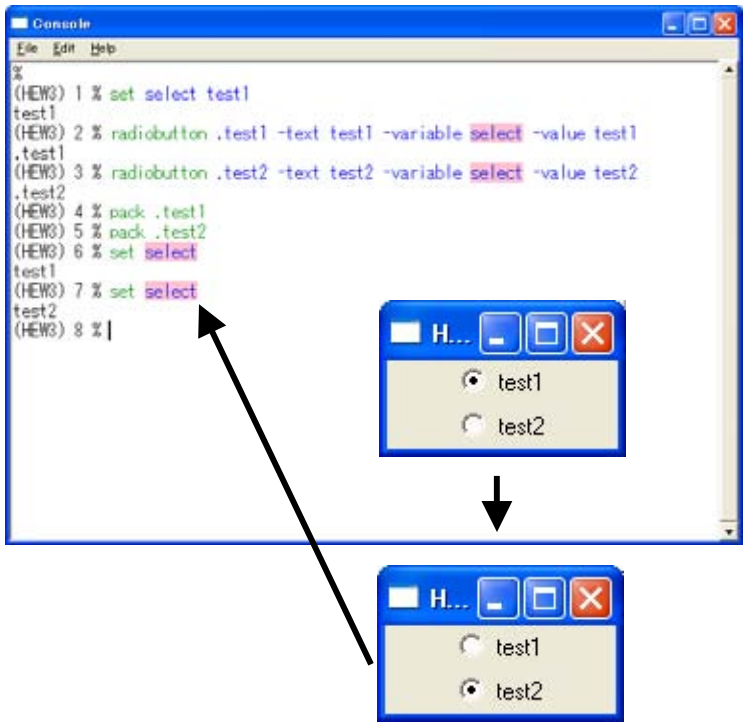
サンプルでは、-text オプションで指定した“ test“ という名前のボタンを作成しています。
また、チェックボタンに初期値を与える為に本サンプルではチェックボタンのパス名に使用している変数 test に set コマンドを用いて 1 を与えています。

2-4-3、ラジオボタンの作成

Tk では、radiobutton コマンドを使用する事でラジオボタンを作成する事ができます。

```
set select test1                    ;#test1 が選択されている事を初期値にします。
radiobutton .test1 –text test1 –variable select –valu test1
radiobutton .test2 –text test2 –variable select –vale test2

pack .test1
pack .test2
```



ラジオボタンの test2 を選択すると -varible オプションで指定した select の値が test2 に置き換えられます。

図　ラジオボタンの作成

ラジオボタンは、 - variable で指定した変数 select にチェックされたボタンの-value 値が渡されます。サンプルプログラムでは、ボタン作成時に変数 select に test1 を設定していますのでラジオボタンの test1 に初期値としてチェックがついています。
また、作成する複数のラジオボタンで-variable オプションの指定する変数名を同じにする事によってボタンのチェックを排他的に動作させる事ができます。

2-4-4、ラベルの作成

Tk では、label コマンドを使用する事で GUI 上に１行のメッセージを表示できます。

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

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

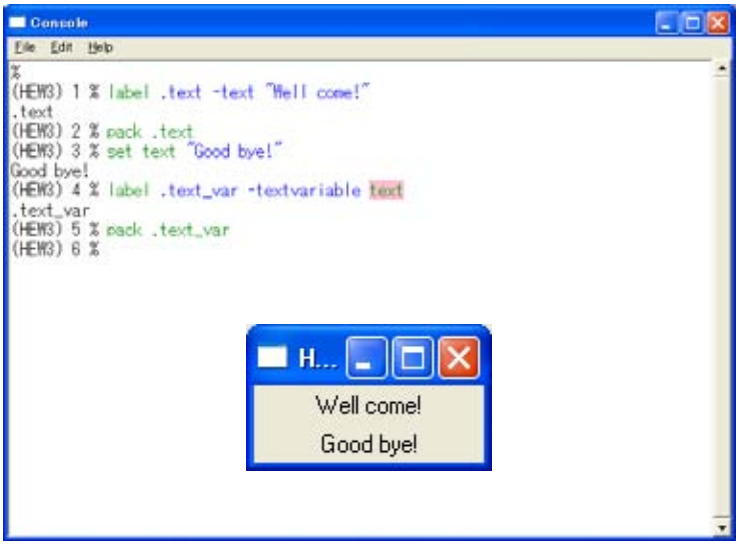


図　ラベルの作成

label コマンドの‐text オプションで指定した文字列が GUI 上に表示されます。
また、予め set コマンドで変数 test に設定しておいた文字列を‐textvariable オプションで指定する事によって表示する事も可能です。

2-4-5、メッセージの作成

Tk では、message コマンドを使用する事で複数行のメッセージを表示することができます。

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

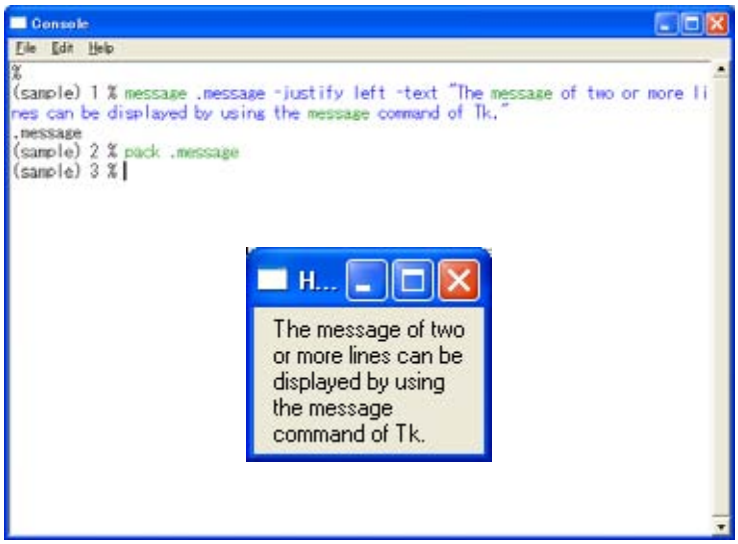


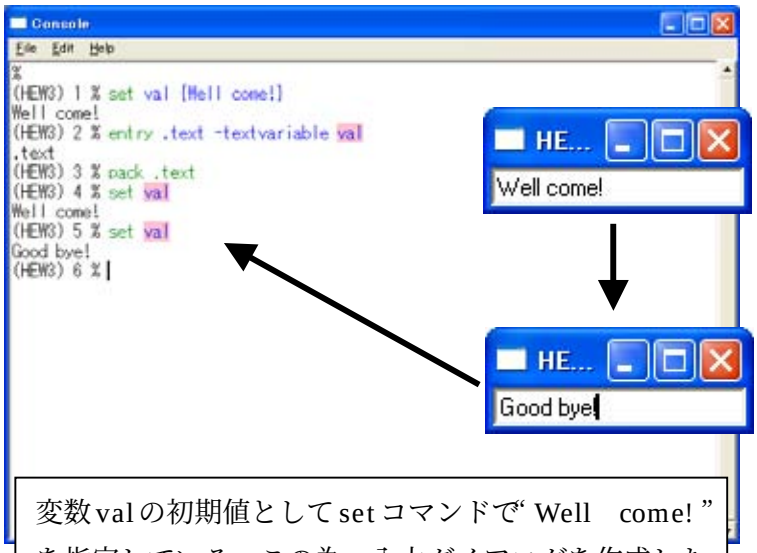
図　メッセージの作成

label コマンドでは一行の出力形式であったのに対して、message コマンドでは複数行にまたがる文字列を表示することが可能です。
また、コマンド入力時にメッセージが長く一行に書ききれない場合は、“ ¥ ” で改行し、複数行に入力する事が可能です。

2-4-6、エントリーの作成

Tk では、entry コマンドを作成する事で入力ダイアログを作成する事ができます。

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



変数 val の初期値として set コマンドで“ Well　come! ”を指定している。この為、入力ダイアログを作成した段階では、“ Well come! ”が表示されます。
入力ダイアログに Good bye!を入力すると、変数 val に入力した文字列が反映されます。

図　エントリーの作成

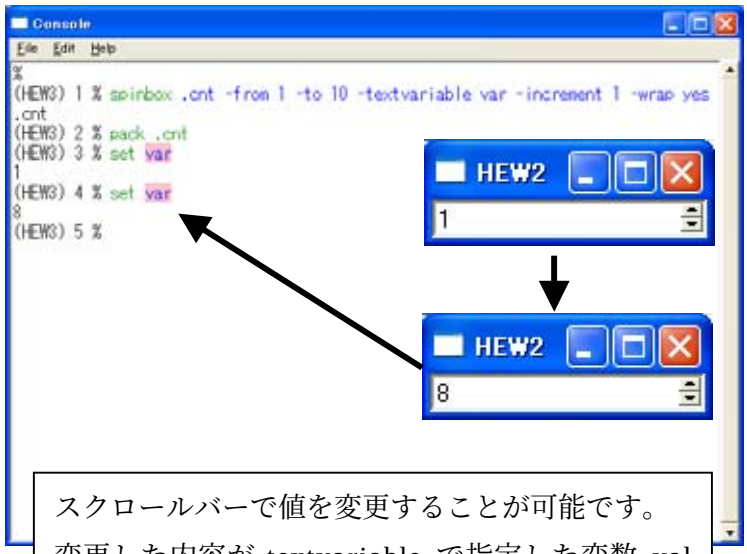
entry コマンドによって作成された入力ダイアログには、1 行の文字列を入力する事が可能です。入力した文字列は、-textvariable オプションで指定している変数 val に反映されます。

複数行の入力ダイアログを作成する場合は、Tk が提供している text コマンドを御使用下さい。text コマンドを使用ずる事で複数行にまたがる入力ダイアログが作成する事ができます。また、text コマンドには、数多くのオプションが用意されています。オプションを使用する事で簡単なエディタを作成することも可能です。

2-4-7、スピンボックスの作成

Tk では、spinbox コマンドを使用する事でスクロールバーによって変更できるスピンボックスを作成する事ができます。

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



スクロールバーで値を変更することが可能です。
変更した内容が-textvariable で指定した変数 val に反映された事が確認できます。

図　スピンボックスの作成

spinbox コマンドでは、-from と-to オプションにてスピンボックスにて設定可能な数値の範囲を指定します。-increment では、スクロールバーにて値を変更する際の刻み幅を指定します。これにより作成されたスピンボックスのスクロールバーを動かす事によって値の変更が出来、変更した内容が-textvariable で指定した変数 val に反映されます。

また、-wrap オプションで yes を指定した場合はスクロールバーの上下で変更する値が-from と-to オプションで指定した範囲を巡回します。(例　0 →・・→ 10 → 0 →・・→ 5)

2-4-8、フレームの作成

Tk では、frame コマンドを使用する事でフレームを作成する事ができます。

```
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
```

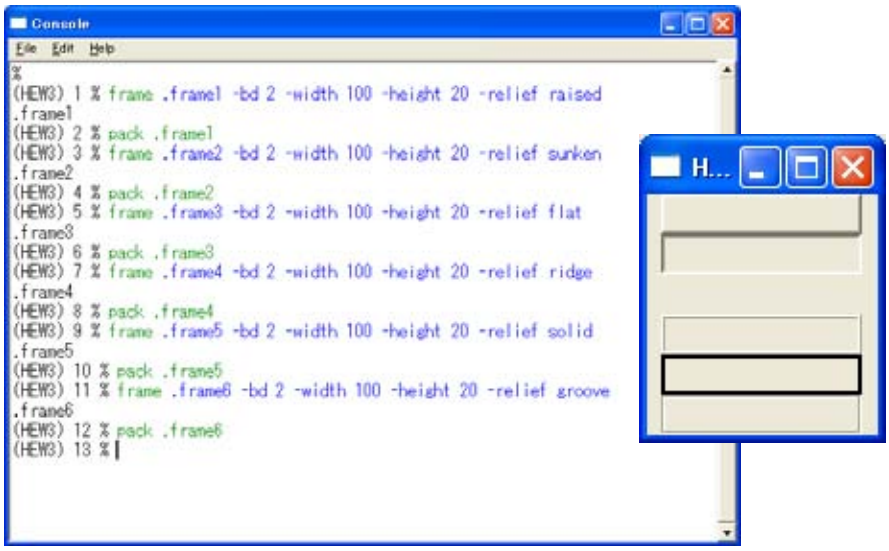


図　フレームの作成

frame コマンドを使用することによってユーザが作成する GUI 環境のレイアウトを整える事が可能です。フレーム内への wideg の配置に関しては別途ご説明します。

frame コマンドでは、-width と-height オプションによって作成するフレームの大きさを指定します。また、-relief オプションの設定により作成されるフレームの形状を変えることができます。

2-4-9、ラベルフレームの作成

Tk では、labelframe コマンドを使用する事でラベルフレームを作成する事ができます。

```
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
```

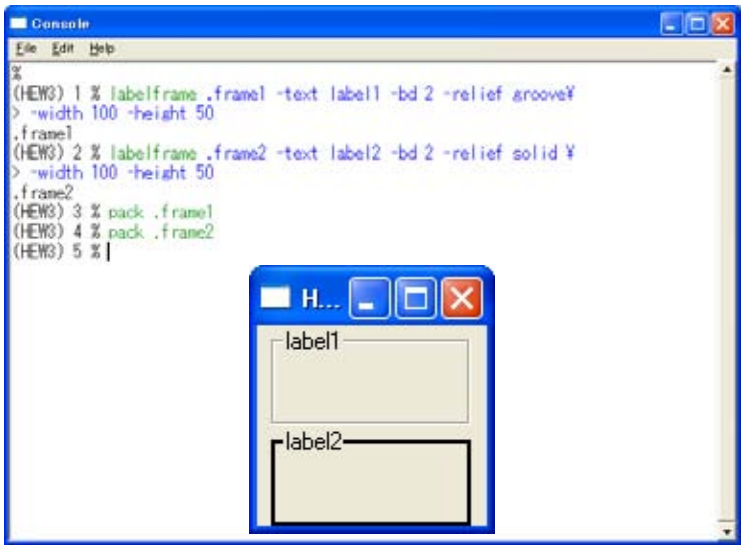


図　ラベルフレームの作成

labelframe コマンドは frame コマンドと同様に、labelframe コマンドを使用することによってユーザが作成する GUI 環境のレイアウトを整える事が可能です。Frame コマンドとの違いは、作成したフレームに-text オプションで指定したラベル名を付加する事ができます。

また frame コマンドと同様に-width と-height オプションによって作成するフレームの大きさを指定します。また、-relief オプションの設定により作成されるフレームの形状を変えることができます。

2-4-10、フレーム、ラベルフレーム内に widget を配置

2-4-9、2-4-10 で作成したフレーム、ラベルフレーム内に Tk の widget を配置する事ができます。

```
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
```

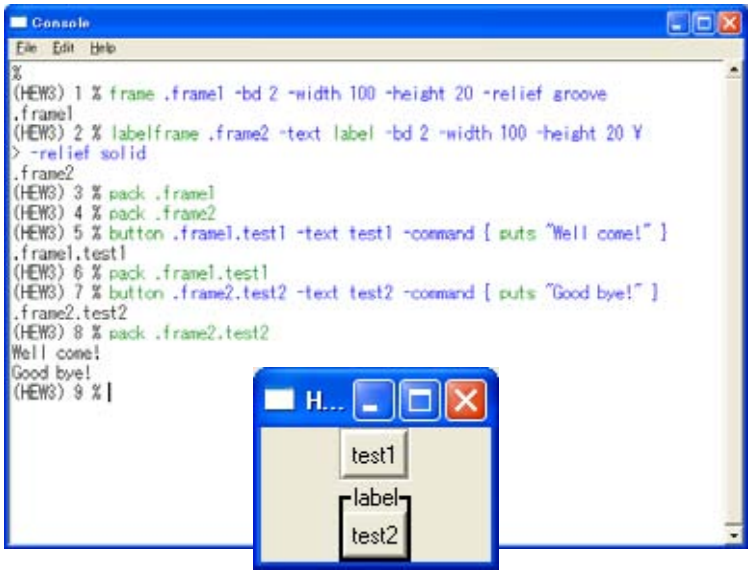


図　widget の配置例

作成したフレームやラベルフレーム内に Tk の widget を配置する場合は、配置する widget のパス名が通常の配置と異なります。

サンプルのプログラムでは、作成したフレームやラベルフレームにボタンを配置しています。フレームのパス名を.frame1、ラベルフレームのパス名を frame2 としているため、配置するボタンのパス名はそれぞれ.frame1.test1、.frame2.test2 と指定します。
これは、フレーム.frame1 にボタン.test1 をラベルフレーム.frame2 にボタン.test2 を配置する事を意味します。

またフレームやラベルフレームのパス名を使用せずに Tk の widget のパス名を指定した場合は、フレームやラベルフレームの外側に widget が配置されます。

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

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

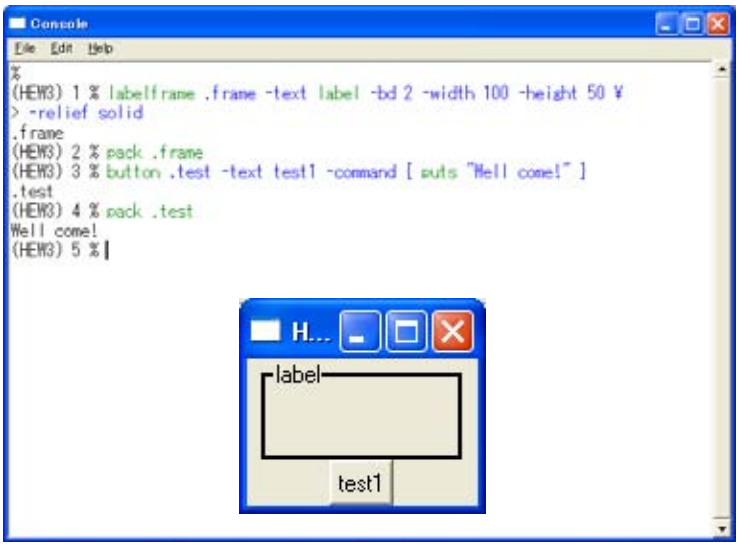


図　widget の配置例

補足

Tcl/Tk のパス名で “ . ” は、ルートを表します。ルートに対してフレームやボタンを配置していく事により作成した GUI 環境のパス名が階層構造になります。

例

パス名	.top	GUI 環境の top
	.top.frame1	top に配置した frame1 のパス名
	.top.frame2	top に配置した frame2 のパス名
	.top.frame2.subframe	frame3 に配置した subframe のパス名
	・　・・	

2-4-11、新規トップレベルウィンドウの作成

Tk では、toplevel コマンドを使用する事で新規の Top レベルウィンドウを作成する事ができます。

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

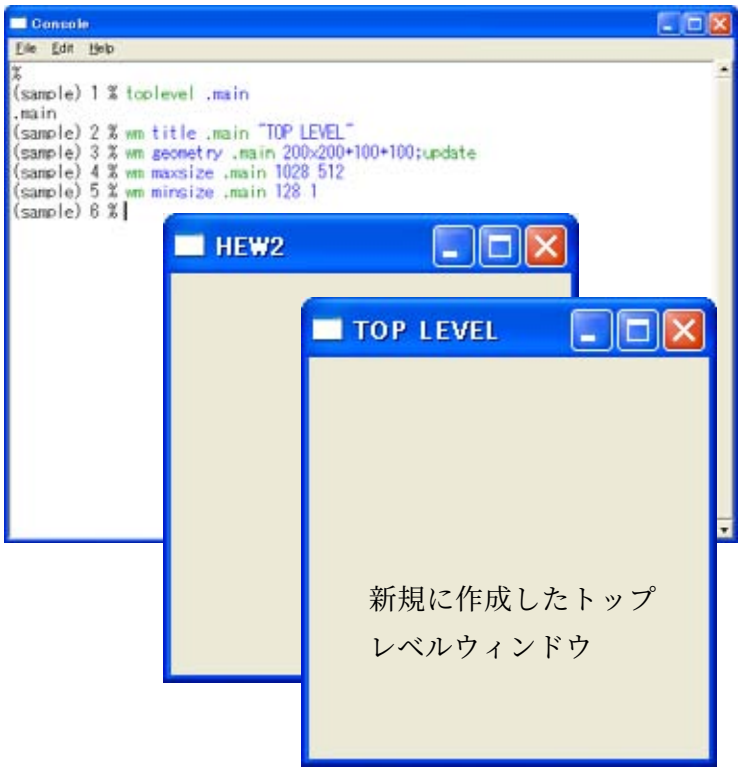


図　トップレベルウィンドウの作成

本サンプルプログラムでは、HEW が準備しているデフォルトのウィンドウ以外に新規にウィンドウを作成して表示します。作成したウィンドウのタイトルやサイズを指定する為に wm コマンドを使用して設定しています。
新規のウィンドウに Tk の widget を配置していく場合は、パス名のはじめが.main と指定する必要があります。

また作成したウィンドウは、destroy コマンドを使用（destroy .path 名）して削除する事が可能です。

2-4-12、メニューの作成

Tk では、menu コマンドを使用する事でツールバーメニューを作成することができます。

```
menu .menu       ;#menu のパス名を設定
.menu add cascade –label file –menu .menu.file
.menu add cascade –label edit –menu .menu.edit
.menu add cascade –label view –menu .menu.view
                 #.menu に追加するカスケード（file、edit、view）を設定します
menu .menu.file –tearoff no
                 # -tearoff のオプションを真にすると作成したメニューを
                 #ウィンドウから切り離す事ができます。
.menu.file add command –label exit –command exit
                 #menu.file に submenu として exit を配置します
                 #選択時のアクションとして exit を定義します
. configure –menu .menu
```

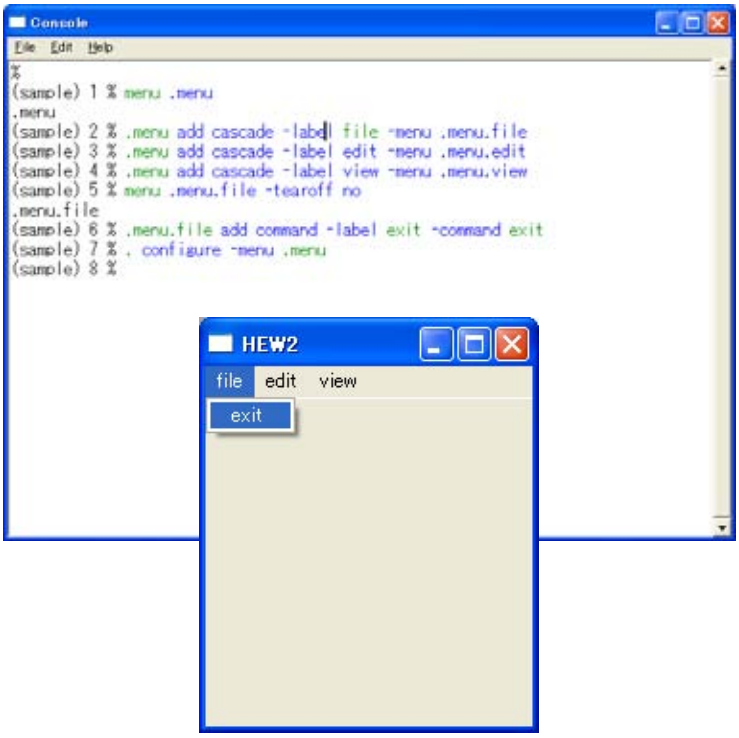


図　メニューの作成

menu コマンドでは、toplevel のウィンドウにプルダウン（カスケード）タイプのメニューを作成することができます。

2-4-13、メニューボタンの作成

Tk では、menubutton コマンドを使用する事でメニューボタンを作成する事ができます。

```
frame .menutop              ;#ツールバーに配置する為に frame コマンドを使用します
pack .menutop –side top –fill x
                            #–side top で作成したウィンドウを top に配置しています
menubutton .menutop.file –text file –menu .menutop.file.menu
                            #-menu で submenu として .menutop.file.menu を配置します
menubutton .menutop.edit –text edit
menubutton .menutop.view –text view
pack .menutop.file .menutop.edit .menutop.view –side left
                            #作成した menubutton を配置します
menu .menutop.file.menu –tearoff 0
                            # -tearoff の オプションを真にすると作成したメニューを
                            #ウィンドウから切り離す事ができます。
.menutop.file.menu add command –label exit –command exit
                            #menutop.file.menu を選択した際のアクションを定義します
```

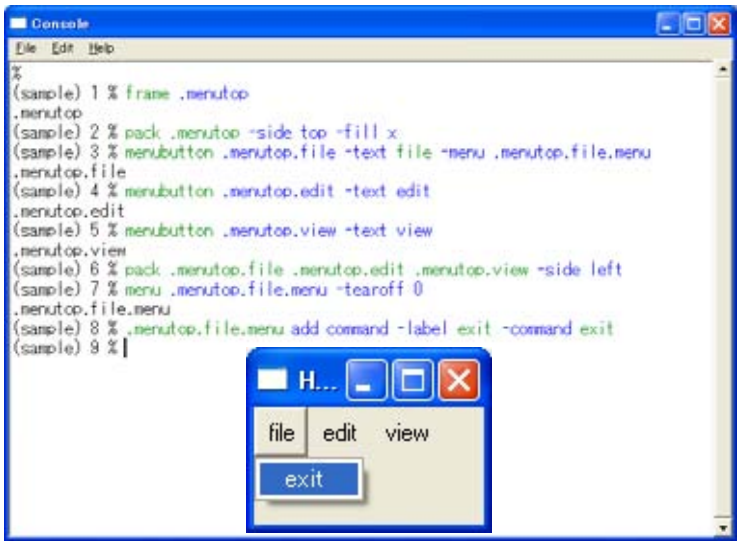


図　メニューボタンの作成

menubutton コマンドで作成したメニューは、機能的には menu コマンドと同じです。Menu コマンドと同じ用にツールバーに配置する場合は、最初に frame コマンドを使用してメニューバー用にフレームを配置する必要があります。

2-4-14、スクロールバーの作成

Tk では、scrollbar コマンドを使用する事で、ウィンドウ等にスクロールバーを付加する事ができます。

```
scrollbar .scroll_h –orient horizontal      ;#横方向のスクロールバーを定義します
scrollbar .scroll_v –orient vertical        ;#縦方向のスクロールバーを定義します
pack .scroll_h                              ;#定義したスクロールバーを配置します
pack .scroll_v –side right
```

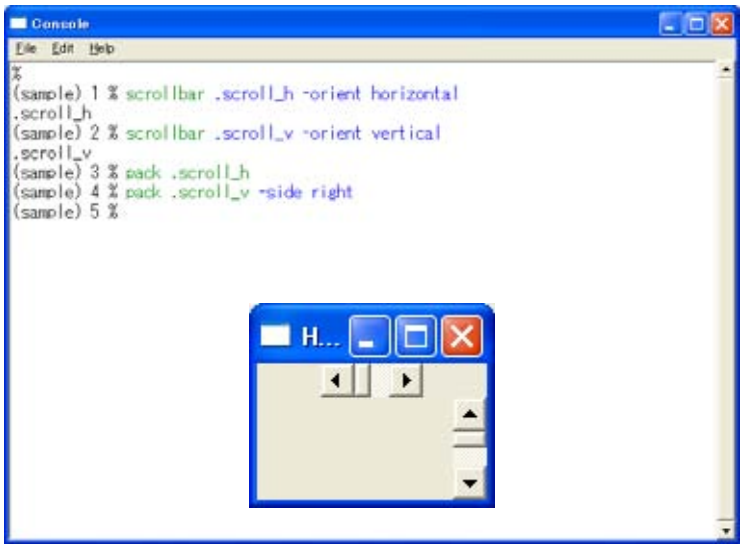


図　スクロールバーの作成

サンプルプログラムでは、トップのウィンドウに配置しています。スクロールバーの定義、配置時のパス指定に別途作成したフレーム等のパス名を考慮して指定すればフレーム等に配置する事も可能です。

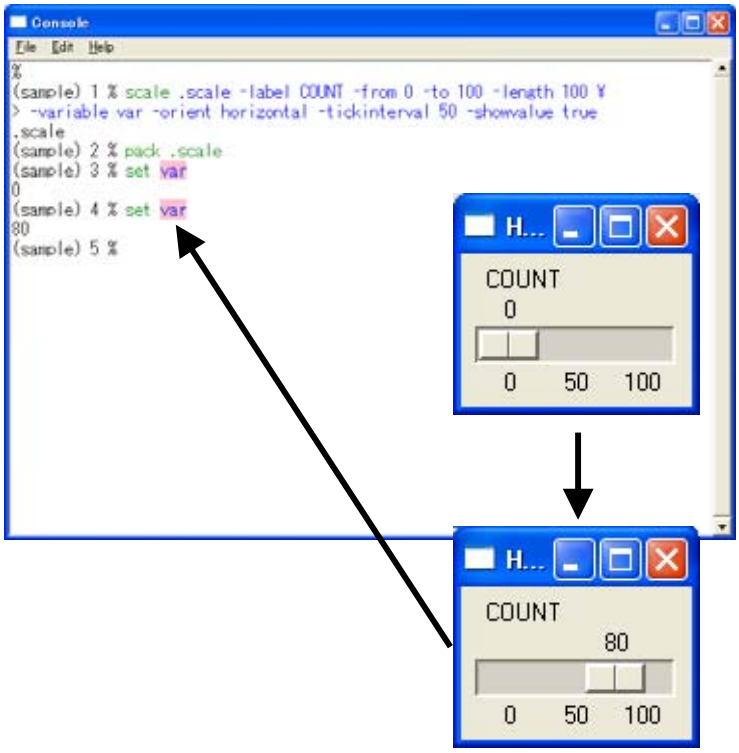
```
labelframe .frame –text label –bd 2 –width 100 –height 50 –relief solid
pack .frame

scrollbar .frame.scroll_v –orient vertical    ;#フレームに配置するためのパス名を指定
pack .frame.scroll_v
```

2-4-15、変数の値を調整するスケールバーの作成

Tk では、scale コマンドを使用する事で変数の値を調整するスケールバーを作成する事ができます。

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



スケールバーの値を変更すると-variable オプションで指定した変数 var の値に反映されます。

図　スケールバーの作成

scale コマンドでは､-from と-to オプションで指定した範囲のスケールバーを作成できます。スケールバーで設定した値は、-variable オプションで指定した変数 var に反映されます。また、-tickinterval オプションでは表示するメモリ間隔の指定、-showvale オプションではスケールバーが指している値の表示を指定しています。

2-4-16、キャンバスの作成

Tk では、canvas コマンドを使用する事で行、テキスト、ポリゴン等を配置したキャンバスを作成する事ができます。

```
canvas .canvas                          ;#作成するキャンバスを定義します
.canvas create oval 10 10 40 40 –fill red –width 3
.canvas create rectangle 50 50 70 70 –fill blue –width 5
pack .canvas                            ;#キャンバスを配置します
```

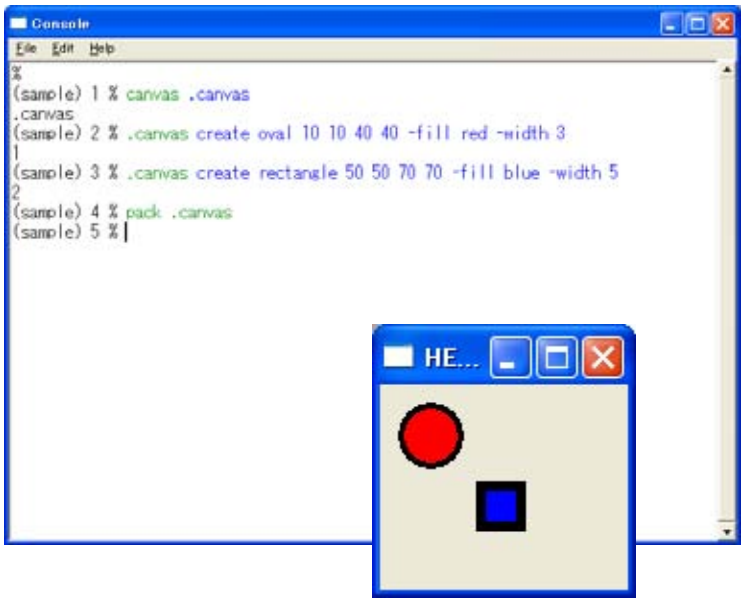


図　キャンバスの作成

2-4-17、オプションメニューの作成

Tk では、tk_optionMenu コマンドを使用する事でオプションメニューを作成する事ができます。

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

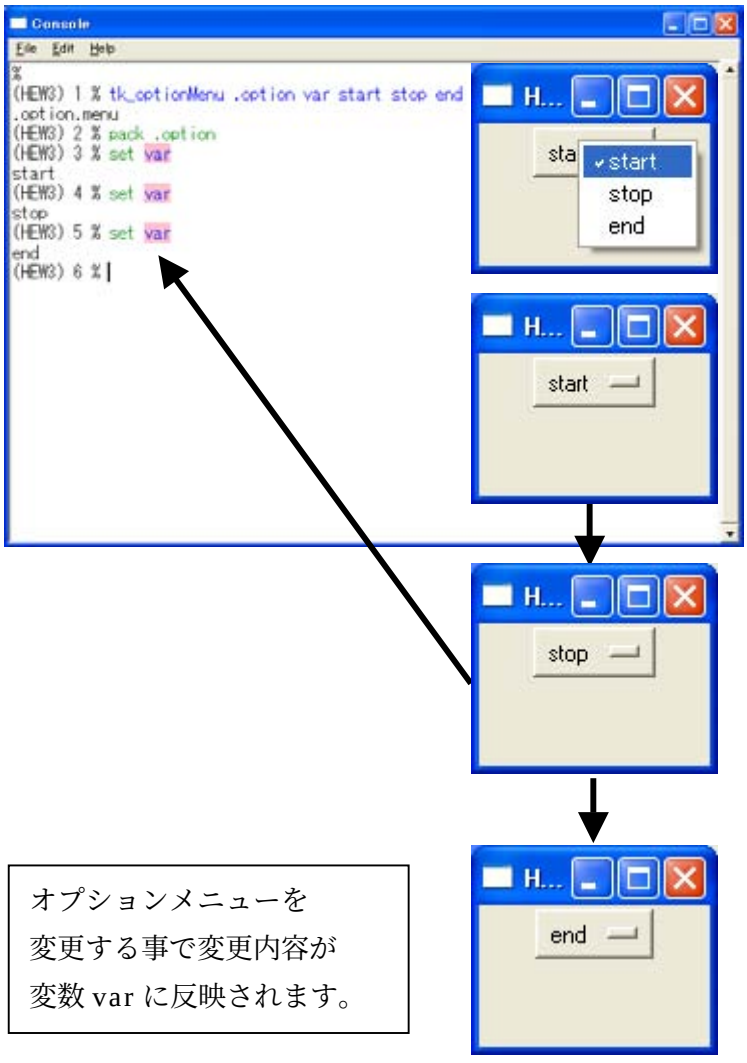


図　オプションメニューの作成

tk_option コマンドでオプションメニューを作成する際に変数 var とオプションメニューに追加するメニュー項目 start、stop、end を指定します。変数 var には、オプションメニューで指定した項目が反映されます。

2-4-18、ポップアップメニューの作成

Tk は、tk_popup コマンドを使用する事でポップアップメニューを作成する事ができます。

```
menu .popupmenu –tearoff no
        # -tearoff yes にするとポップアップメニューをウィンドウから切り出せます
.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
        #メニューを選択した際のアクションを-command オプションで指定可能です
bind . <3> { tk_popup .popupmenu %X %Y }
        #位置を指定する場合は、%X と%Y に数値で指定します
```

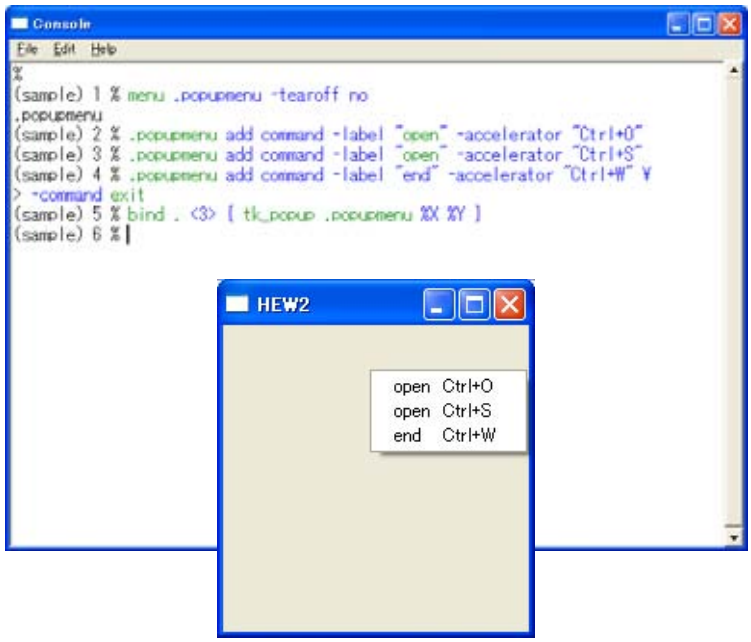


図　ポップアップメニューの作成

menu として作成した.popupmenu を tk_popup コマンドでポップアップメニューとして定義しています。定義したメニューをウィンドウの任意の箇所で開く為に

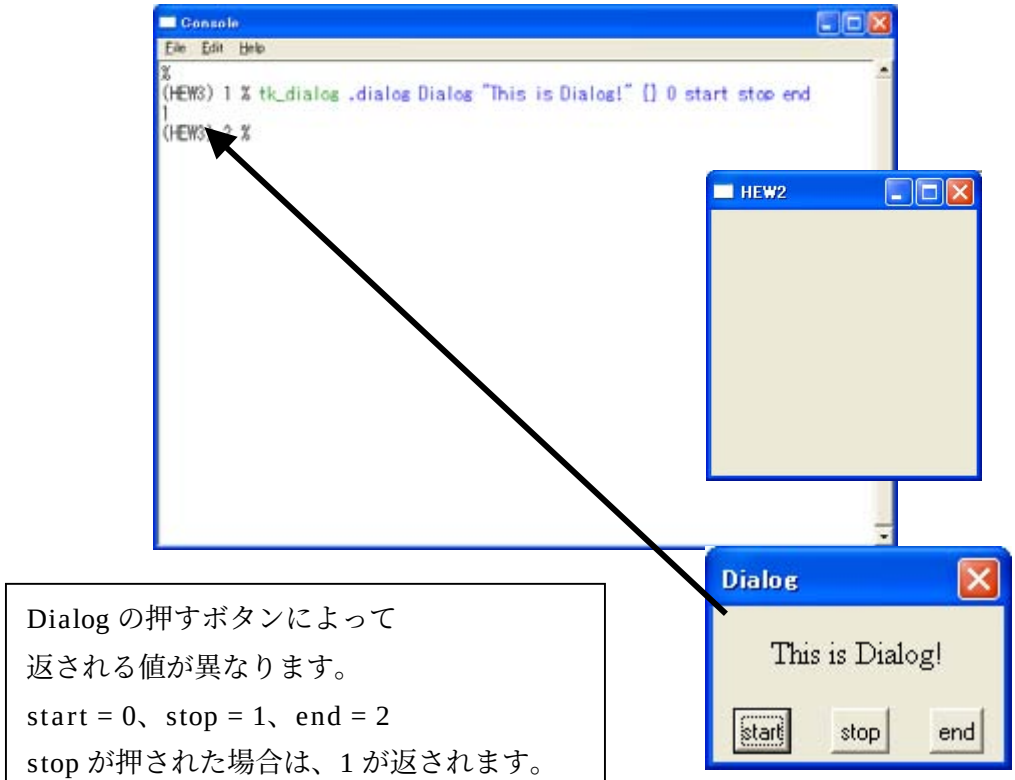
bind . <3> { tk_popup .popupmenu %X %Y }

で設定しています。これによりポップアップメニューをウィンドウの任意の位置(%X,%Y)で開くことが可能です。位置を指定する場合は、%X と%Y に値で指定します。

2-4-19、簡単なダイアログの作成

Tk では、tk_dialog コマンドを使用する事で簡単なダイアログを作成する事ができます。

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



Dialog の押すボタンによって
返される値が異なります。
start = 0、stop = 1、end = 2
stop が押された場合は、1 が返されます。

図　ダイアログの作成

tk_dialog コマンドでダイアログを作成する際、Windonws のタイトルに表示する Dialog と Windows 内に表示するメッセージ This is Dialog!を指定する事ができます。
また、ダイアログ内に表示するボタンの初期値 0 とボタン start、stop、end を指定する事でダイアログが作成されます。

ダイアログ内の start、stop、end をユーザが押す事で結果としてそれぞれ 0、1、2 の値が返されます。

2-4-20、メッセージダイアログの作成

Tk では、tk_messageBox コマンドを使用する事でメッセージダイアログを作成することができます。

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

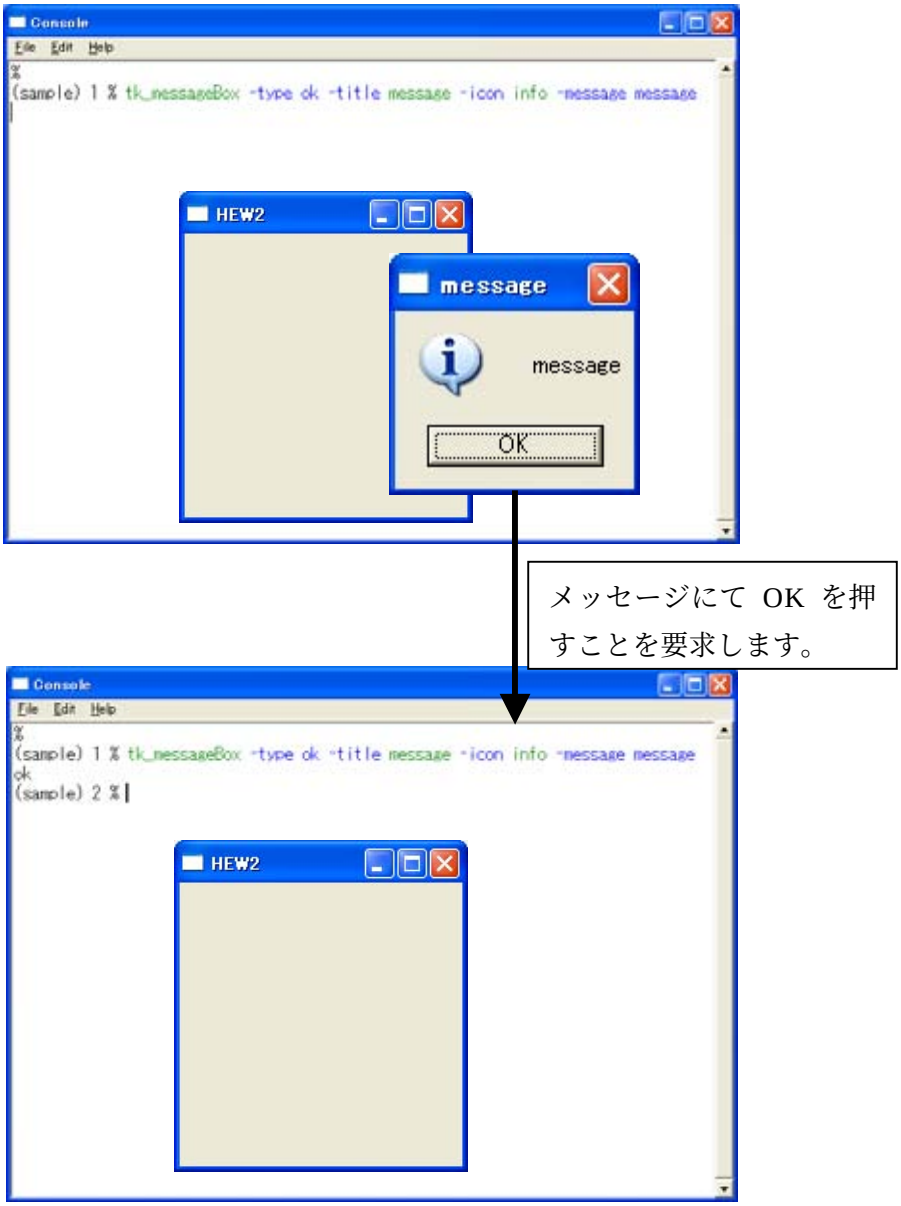


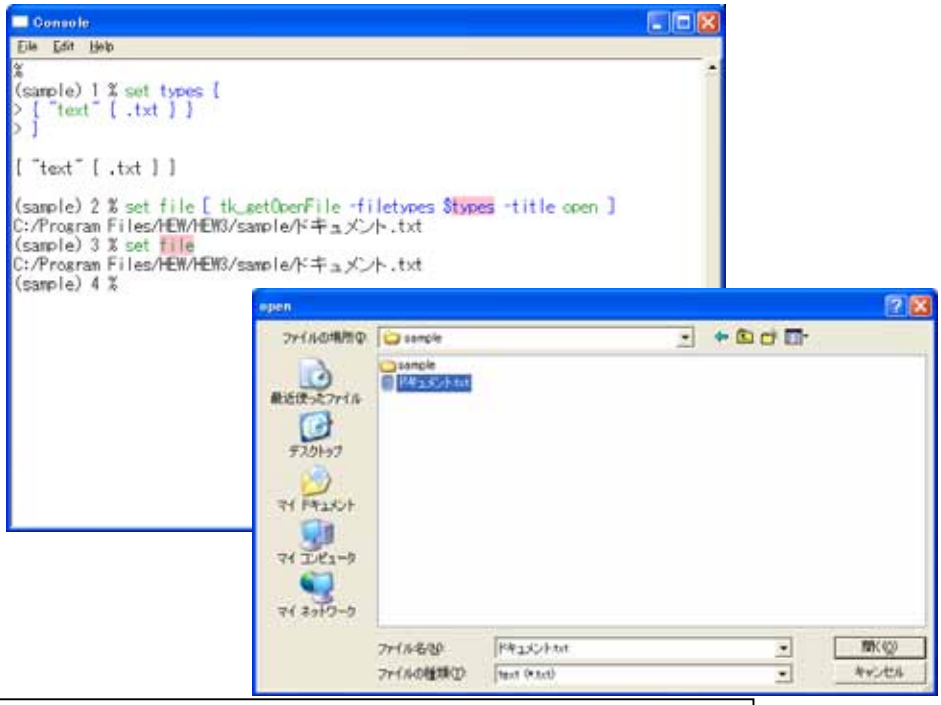
図　メッセージダイアログの作成

tk_messageBox コマンドでは、ユーザに確認を促すウィンドウを作成する事ができます。-type オプションでボタンの type、-title オプションでウィンドウのタイトル、-message オプションでウィンドウ内に表示するメッセージを指定する事ができます。

2-4-21、既存ファイルを開くダイアログの作成

Tk では、tk_getOpenFile コマンドを使用する事で既存ファイルを開くダイアログを作成する事ができます。

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



ファイルを選択してファイルを開くと file 変数には指定したファイルの格納先情報が反映されます。

図　ファイルオープンダイアログの作成

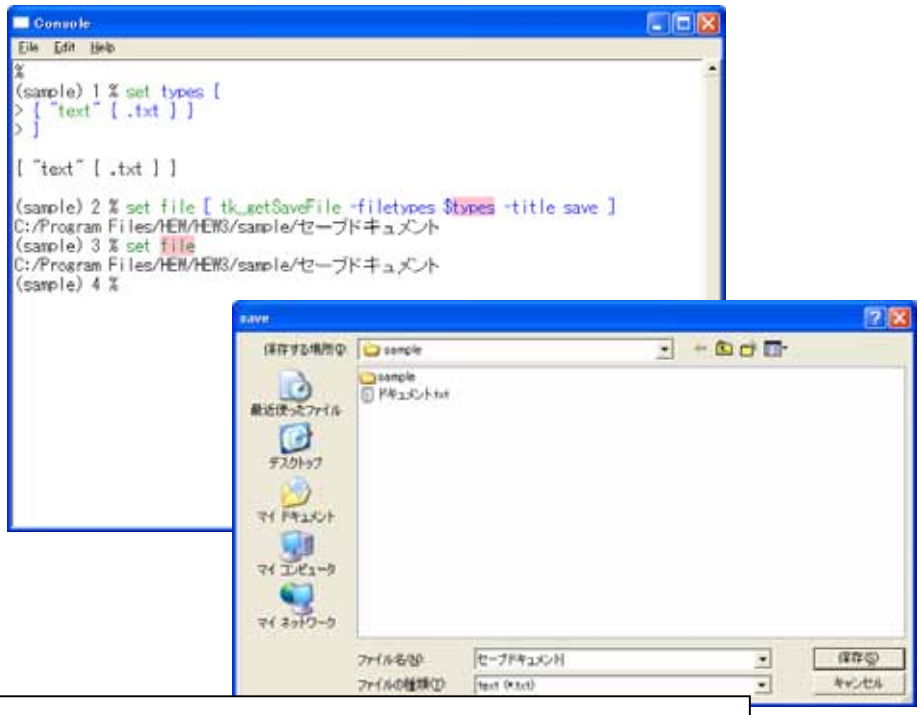
Tk_getOpenFile コマンドで既存ファイルを開くダイアログを作成する際、-filetypes オプションでファイルの種別を指定し-title オプションで window のタイトルを指定できます。サンプルプログラムでは、-filetypes オプションで指定しているファイル種別に types 変数を使用しています。type 変数は、予め set　type{ . . . }　コマンドでファイル種別として text [*.txt]を設定しています。

また、本コマンドの実行結果はファイルの格納先の情報が返されます。サンプルプログラムでは、set コマンドを使用して変数 file に結果を格納しています。

2-4-22、新規ファイルを開くダイアログを作成

Tk では、tk_getSaveFile コマンドを使用する事で新規のファイルを開きセーブするダイアログを作成する事ができる。

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



セーブするファイルを指定してファイルをセーブすると file 変数にセーブしたファイルの格納先情報が反映されます。

図　ファイルセーブダイアログの作成

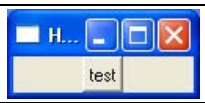
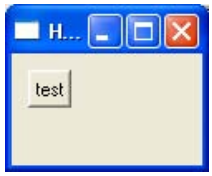
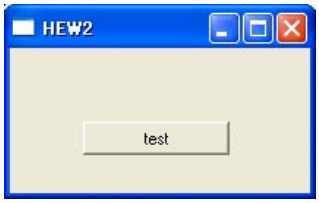
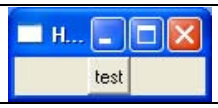
Tk_getSaveFile コマンドでファイルのセーブダイアログを作成する際、Tk_getOpenFile コマンドと同様に-filetypes オプションでファイルの種別、-title で window のタイトルを指定する事が可能です。

また本コマンドの実行結果も Tk_getOpenFile コマンドと同様に、ファイルの格納先が返されます。

2-4-23、widget の配置

ここまでのサンプルプログラムでは、widget の配置方法を pack コマンドで説明してきました。しかし widget を配置する為のコマンドとしては pack、place、grid コマンドがあります。それぞれ配置方法が異なりますので作成する環境に応じて御使用下さい。

表 widget の配置例

コマンド	コマンド使用例	レイアウト例
pack	Widget を方位でレイアウトします。 -side [left,right,top,bottom]等で指定が可能	
	pack [button .test –text test –command { go }]	
	pack [button .test –text test –command { go }] –side left	
place	Widget を座標でレイアウトします。 -x,-y で座標指定、-width で大きさの指定が可能	
	button .test –text test –command { go } place .test –x 10 –y 10	
	button .test –text test –command { go } place .test –x 50 –y 50 –width 100	
grid	Widget を格子上にレイアウトします。 -column,-row で格子位置、-padx,-pady で余白等の指定が可能です。	
	button .test –text test –command { go } grid .test	
	button .test –text test –command { go } grid .test –column 3 –row 4 -padx 3 –pady 5	

ここで指定したオプションは、一例です。オプション設定は、他に数多くありますので Tcl/Tk のリファレンスマニュアル等をご参照下さい。

3、HEW 上での Tcl/Tk プログラミング

HEW 上で Tcl/Tk のプログラミングが可能です。
HEW で使用可能なコマンドを Tcl/Tk で作成した GUI（ボタン等）に割り当て等を行い、ユーザ独自の開発環境を構築していきます。これによりユーザが任意に作成した GUI 環境から HEW に対してコマンドを発行し HEW 上でのシミュレーション制御等が可能な環境を開発できます。

Tcl/Tk で作成可能な環境は、例として次のような環境があげられます。

・Tcl/Tk から HEW へコマンドを発行。
　HEW へコマンドを発行し HEW のシミュレーション実行等を制御
・Tcl/Tk から HEW へコマンドを発行、発行したコマンドに対する結果を受け取る。
　HEW へコマンドを発行し、発行したコマンドに対する実行結果を Tcl/Tk 側で
　受け取り表示

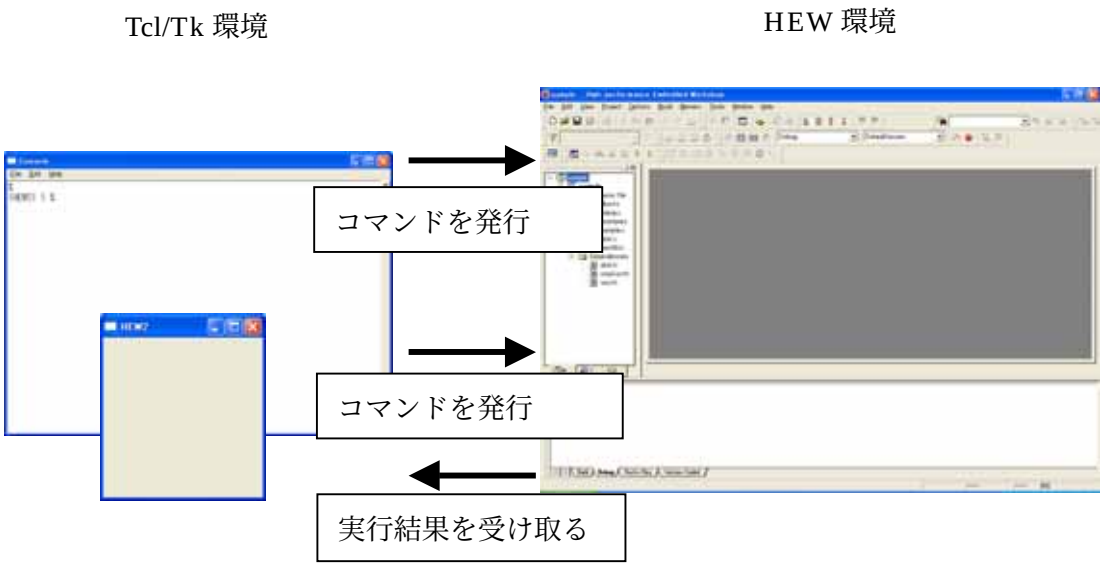


図　HEW と Tcl/Tk の関係

3-1、Tcl/Tk で使用可能な HEW のコマンド

HEW では、HEW のコマンドラインで使用可能な多くのコマンドがあります。
この HEW のコマンド群で Tcl/Tk から使用可能なコマンドを用いてユーザ独自の開発環境を作成して頂く事が可能です。

Tcl/Tk で使用可能な HEW のコマンド一覧は、Tcl/Tk の Console ウィンドウに lis コマンドを入力することでご確認頂けます。表示されたコマンド内容をご確認の上御使用下さい。

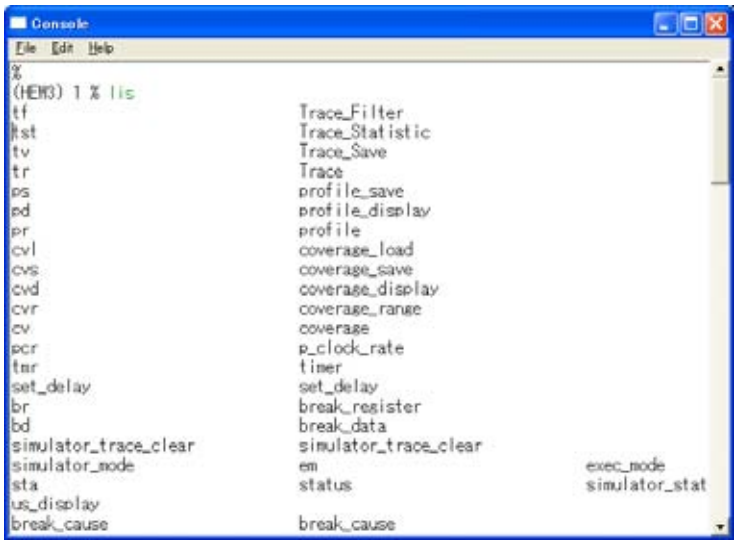


図　lis コマンド実行時の表示例

3-2、HEW シミュレータへの制御コマンド環境を作成

Tcl/Tk で HEW シミュレータへの制御環境の GUI を作成することができます。

```
～　sample プログラム　～
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 }
```

本サンプルプログラムでは、HEW のシミュレーション実行時の最も簡単なコマンドを Tcl/Tk の GUI 上のボタンに割り付けた制御環境を作成しています。

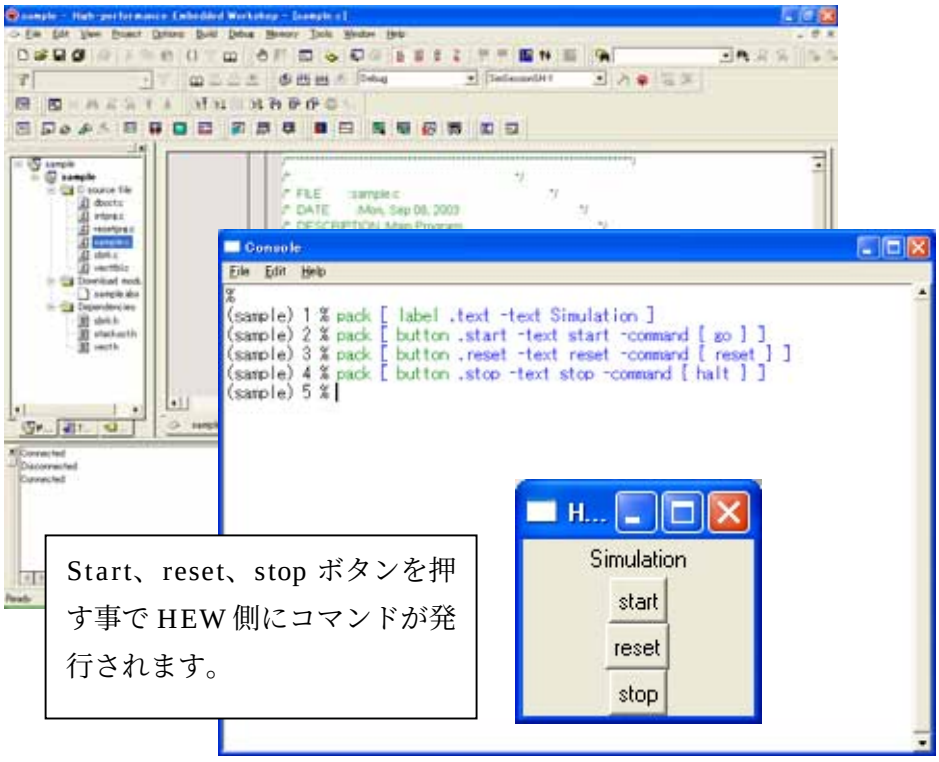


図　実行例

サンプルプログラムでは、Tcl/Tk の label、button コマンドを使用して Tcl/Tk の GUI 環境を作成します。label コマンドでは、window に表示するメッセージ Simulation を指定、button コマンドでは GUI 上に配置するボタンの指定、及びボタンに割り付けるコマンドを -command オプションの“{ }”で括られた箇所に HEW コマンドを指定する事によって指定しています。また、ボタンの配置には pack コマンドを使用して配置しています。
これにより Tcl/Tk の GUI 環境にある“ start ”,“ reset ”,“ stop ”ボタンを押すと HEW の

シミュレーション環境には、それぞれ go、reset、halt のコマンドが発行されます。

3-3、HEW シミュレータへの擬似割り込み入力コマンドを作成

Tcl/Tk で HEW へ割り込み信号の入力を行う環境を作成する事ができます。

```
～　sample　プログラム ～
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 }
```

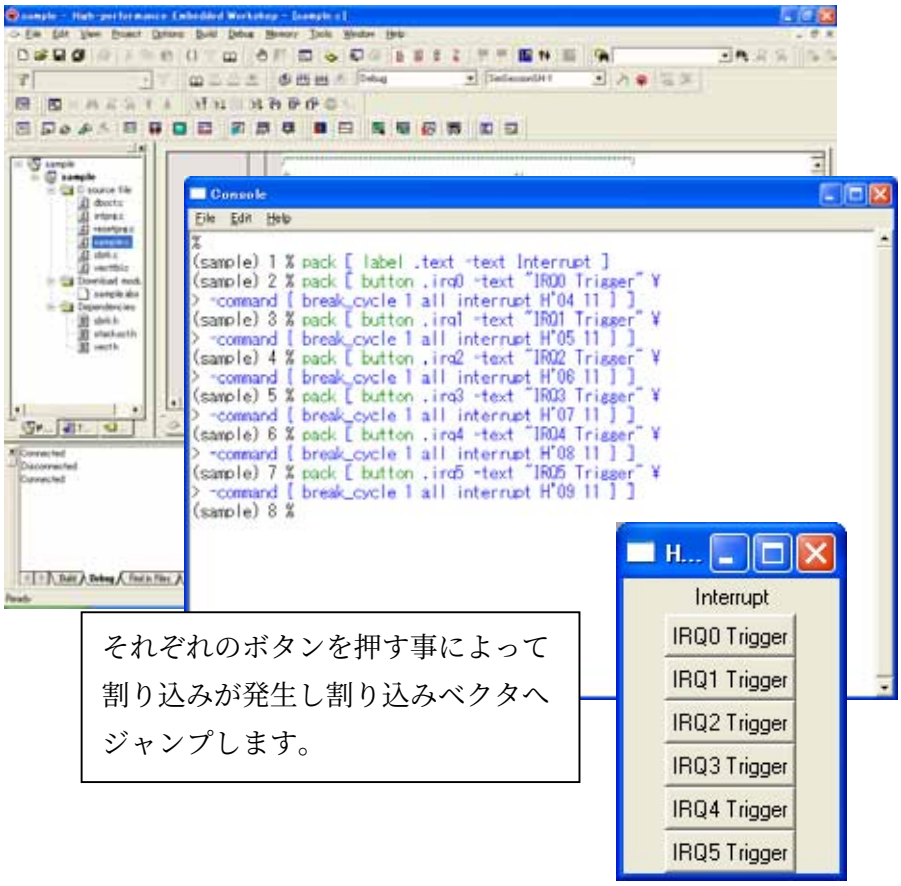


図　実行例

本サンプルプログラムでは、HEW のシミュレータの機能にある擬似割り込み機能を用いています。Tcl/Tk で作成した GUI 上のボタンに擬似割り込みを発生するコマンド break_cycle を指定して、シミュレーション実行時の任意の箇所で割り込みを発生することを可能にしています。

擬似割り込みのは、HEW シミュレータのコマンドにある BREAK コマンドを使用して実現することができます。

表　BREAK コマンド

コマンド	コマンド使用例
break_access	break_access <start_addr> [< end_addr>] [<mode>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_cycle	break_cycle <cycle> [<count>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_data	break_data <addr> <data> [<size>] [<option>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_register	break_register <register> [<data> <size>] [<option>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]
break_point	break_point <addr> [<count>] interrupt <interrupt_type1> <interrupt_type2> [<priority>]

通常のシミュレーション時には、これらのコマンドをシミュレーション実行前に設定しておく必要があります。これにより擬似割り込みを発生させます。

本サンプルプログラムでは、break_cycle 1 all …のコマンドをボタンに割り付けてあります。（CPU：SH1 で設定）これによりシミュレーション実行途中で任意の箇所でボタンが押されるとボタンがおされてから 1cycle シミュレーションが進んだところで擬似割り込みが入力されます。

3-4、シミュレーション制御環境を作成

シミュレーション実行の制御コマンドを構築するスクリプトファイルを作成し、ひとつの統合環境を作成します。

～　sample　プログラム　～

```
#!/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
##############################################################################
# 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
button .top.irq0  -command {break_cycle 1 all Interrupt H'04 11} -pady 0 ¥
       -text {Trigger IRQ0} -width 15
button .top.irq1  -command {break_cycle 1 all Interrupt H'05 11} -pady 0 ¥
       -text {Trigger IRQ1} -width 15
```

新規にトップレベルの
ウィンドウを定義

ボタンの定義

```
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
```

ボタンの配置

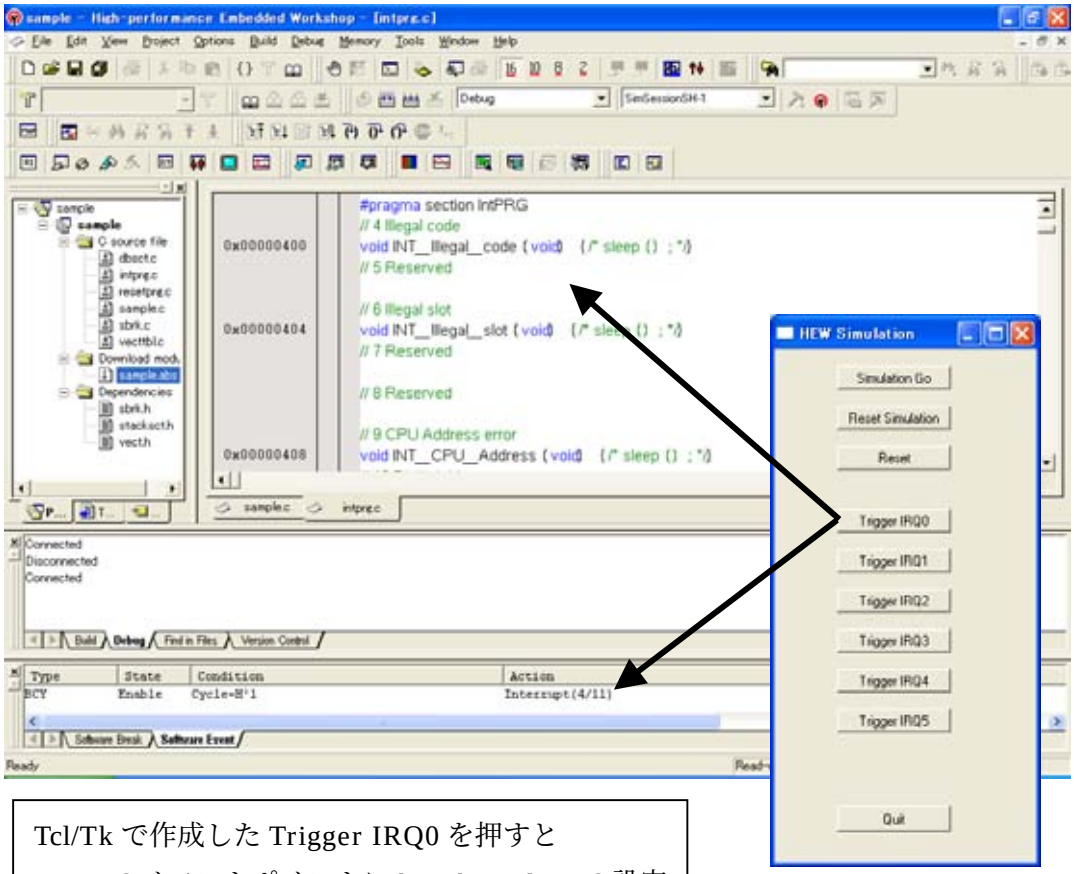
HEW シミュレータの機能を使用して IRQ0～IRQ5 の擬似割り込みを発生させると、HEW のシミュレータには break コマンドの設定が残ります。再度シミュレーションを実行させる為には、この break コマンドを解除する必要があります。

本サンプルプログラムでは、擬似割り込み発生の他に次のコマンドを GUI 上のボタンに設定してあります。

```
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
```

これらのコマンドは、-command オプションの指定“{　}”内に“;”で区切ることによって複数のコマンドを羅列しています。これによりボタンが押される事によって左から順に HEW に対してコマンドが発行されていきます。

本サンプルコマンドでは、go、reset go コマンドの実行前に break_cycle コマンドをクリアするために break_clear コマンドを発行しています。これによりシミュレーションの再実行が可能になります。



Tcl/Tk で作成した Trigger IRQ0 を押すと
HEW のイベントポイントに break_cycle 1 の設定がされ擬似割り込みが入力されます。
プログラムの実行は、割り込みベクタへとジャンプします。

図　実行例

3-5、HEW シミュレーションへの入力制御環境を作成 1

シミュレーション実行時に HEW シミュレータへ値を設定する入力制御環境を作成します。
外部からの入力を考慮した制御環境を作成する事が可能です。

～ sample プログラム ～

```
#!/bin/sh

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

# 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

# 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

# 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
```

変数 addr と data に
初期値を設定

アドレス入力用の
entry ボックスを作成

Label と entry の配置には、
place コマンドで座標位置を指
定して配置

データ入力用の
entry ボックスを作成

設定したアドレス、データを
HEW のコマンドを使用して設定
変数 addr、data は、entry コマ
ンドで設定された値を使用

本サンプルプログラムでは、entry ボックスを使用して任意のアドレス、データを HEW に設定する事が可能な入力制御環境を作成しています。アドレスとデータを GUI 環境に 16 進数で入力し set data ボタンを押すと HEW のメモリ空間にデータが設定されます。

HEW へのデータ設定は、memory_fill コマンドを使用して設定しています。Button コマンドの-command オプションによってボタンが押されたときに HEW に発行する memory_fill コマンドを指定しています。memory_fill コマンドで渡すアドレス値とデータ値は、entry コマンドによって設定された変数 addr と data を参照（データ参照時には、$を使用）することで渡しています。

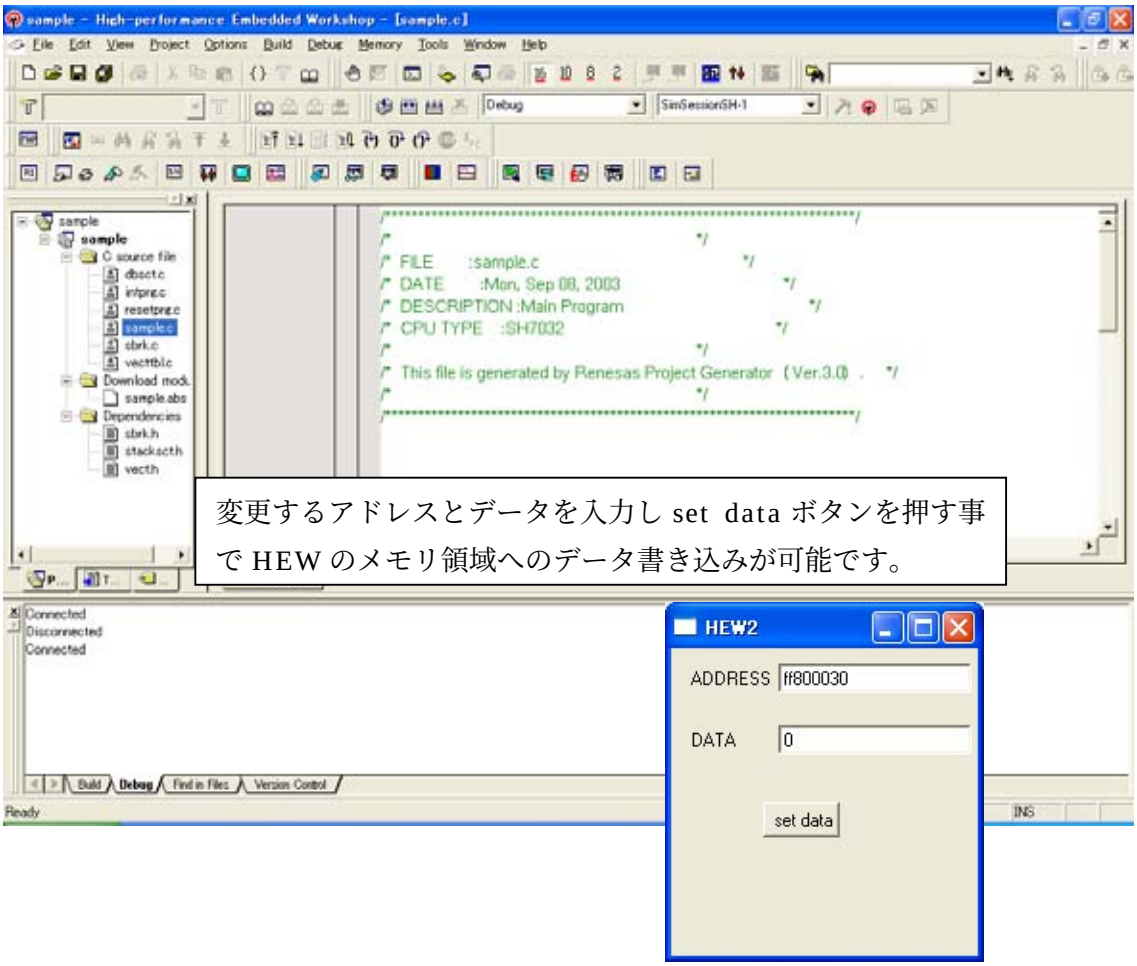


図　実行例

3-6、HEW シミュレーションへの入力制御環境を作成 2

Tcl/Tk 上から任意の数値を HEW のメモリ空間やレジスタへ設定する環境を構築します。

```
～　sample　プログラム　～
wm geometry . 350x280

# 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

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

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

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

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
```

既存 window サイズの変更

データ設定用の scale バーを定義

アドレス設定用の entry ボックスを定義

データ設定用のラジオボタンを定義

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

# 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
        }
}

# 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"
        }
}
```

各変数の初期値を設定

データサイズの設定により HEW へ設定するデータを加工するプロシジャを定義

データサイズの設定により HEW へ発行するコマンドの引数（データサイズ）を生成するプロシジャを定義

```
# 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 ]
        }
}
```

データサイズの設定により HEW へ発行するコマンドの引数（先頭アドレス）を生成するプロシジャを定義

```
# 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 ]
        }
}
```

データサイズの設定により HEW へ発行するコマンドの引数（終了アドレス）を生成するプロシジャを定義

```
# 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
```

HEW へデータを設定するボタンを定義
ボタンには、各プロシジャの呼び出しと HEW への memory_fill コマンドの発行をコマンドとして設定

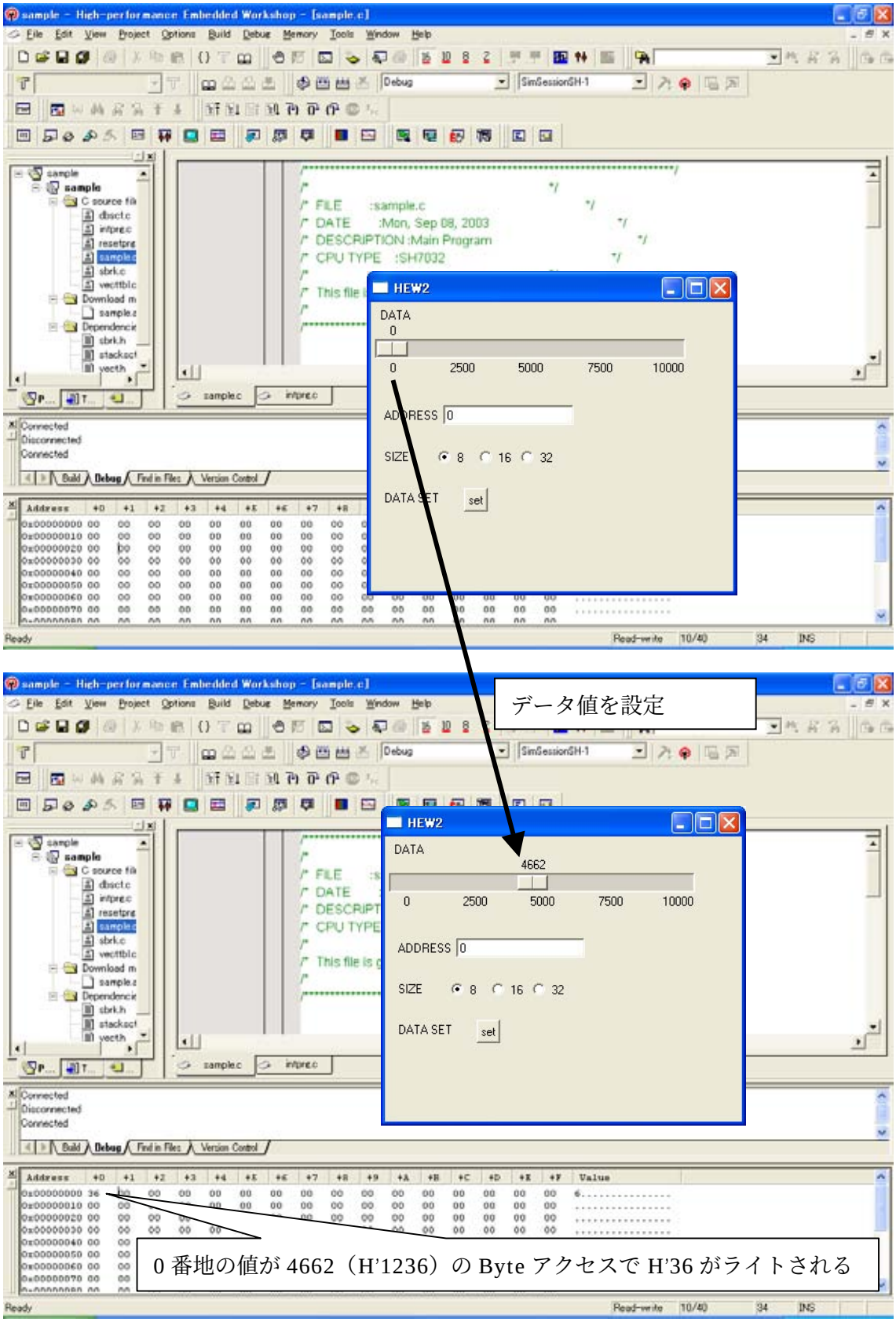

HEW2
DATA
0
0
2500
5000
7500
10000
ADDRESS
0
SIZE
8
16
32
DATA SET
set
データ値を設定
4662
0 番地の値が 4662（H'1236）の Byte アクセスで H'36 がライトされる

図　実行例

本サンプルでは、10 進数（アナログ値）で値を指定し、アドレスとデータサイズを選択した後に set ボタンを押すと指定したアドレス値にデータが格納されます。

0 番地に対して SIZE8 を選択し BYTE で 4662 をライトした場合は、下位 8bit の 36 が格納されます。また、0 番地に対して SIZE 16 を選択し WORD で 4662 をライトした場合は H'1236 が、SIZE32 を選択し LONG で 4662 をライトした場合は、H'00001236 がそれぞれ格納されます。

サンプルプログラムでは、データ、アドレス、データサイズの設定と設定した値を HEW に設定するボタンで構成されています。

また、proc コマンドを使用してデータ、開始アドレス、終了アドレスの変数を加工するプロシジャを定義しています。このプロシジャを button コマンドの-command オプションで使用する事で各データを設定した後に set ボタンを押す事で HEW へのデータ設定時に必要な引数を生成しています。

各プロシジャは、引数の受け渡しを行わない簡単な構成で作成してあります。プロシジャでは、Tcl/Tk のプログラム内で使用している変数を global 宣言する事で参照可能にしています。

3-7、HEW シミュレーションからの出力制御

HEW シミュレーションへの入力制御は、Tcl/Tk から HEW への一方通行のコマンド発行のため Tcl/Tk で HEW へのコマンド発行を行う極簡単なスクリプトを作成する事で実現が可能でした。

しかし、HEW シミュレーションからの出力制御では、Tcl/Tk から HEW へコマンドを発行した結果を受け取る際に変数を介して直接データの受け渡しが行えません。

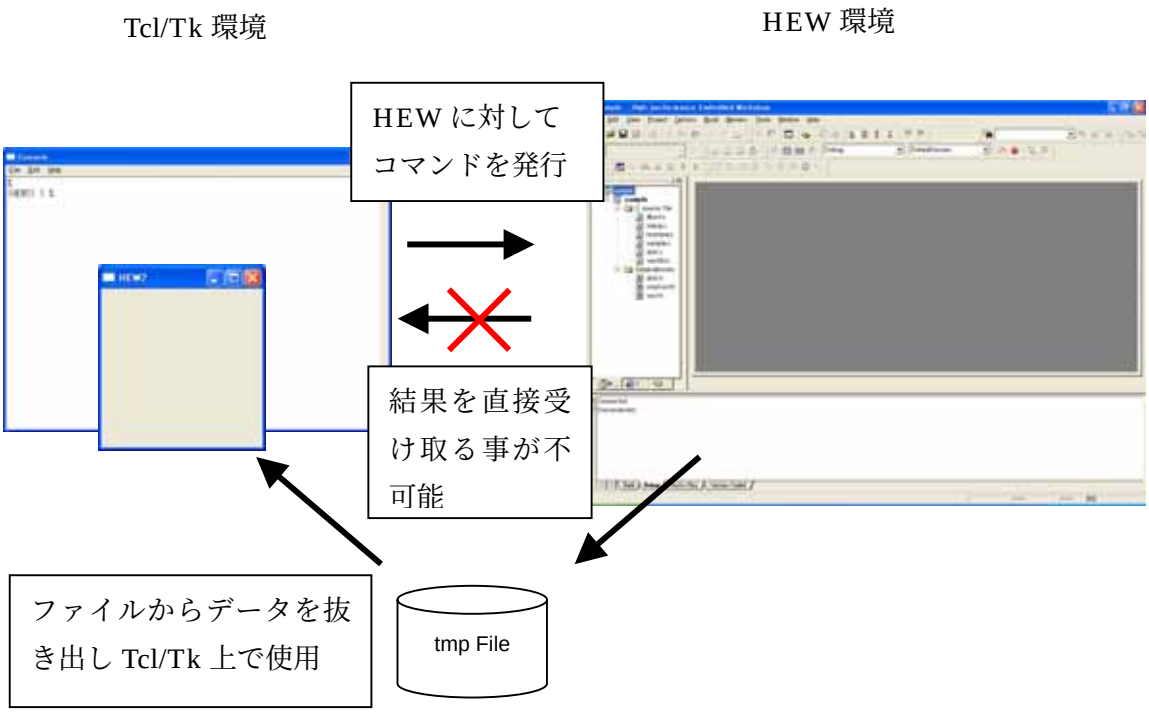


図　HEW シミュレータへの出力制御

対処方法としては、発行したコマンドに対する出力結果を tmp.file を作成して一時的にデータを格納します。Tcl/Tk では、このファイルに格納された内容を参照して HEW から結果としてデータを受け取ります。

3-8、HEW シミュレーションからの出力制御環境を作成

シミュレーション実行時に HEW シミュレータから値を取得する出力制御環境を作成します。メモリやレジスタの値を取得し Tcl/Tk への出力を考慮した制御環境を作成する事が可能です。

```
～ sample プログラム ～
set content "Display Address "

destroy .b
button .b -text "Update" -command  {
                                        ファイルのオープンチェック
                # 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]   tmp ファイルのデータを取り込む
                         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
```

append コマンドは、文字列 b に[lindex コマンドは d の j 番目の要素　を取得]で取得した文字列を追加します。

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

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

destroy .labelcontent
label .labelcontent -text $content

place .labelcontent -x 90 -y 10

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

表示を更新するために destroy コマンドで
前回の表示結果を破棄した後、結果を表示

ボタンを配置

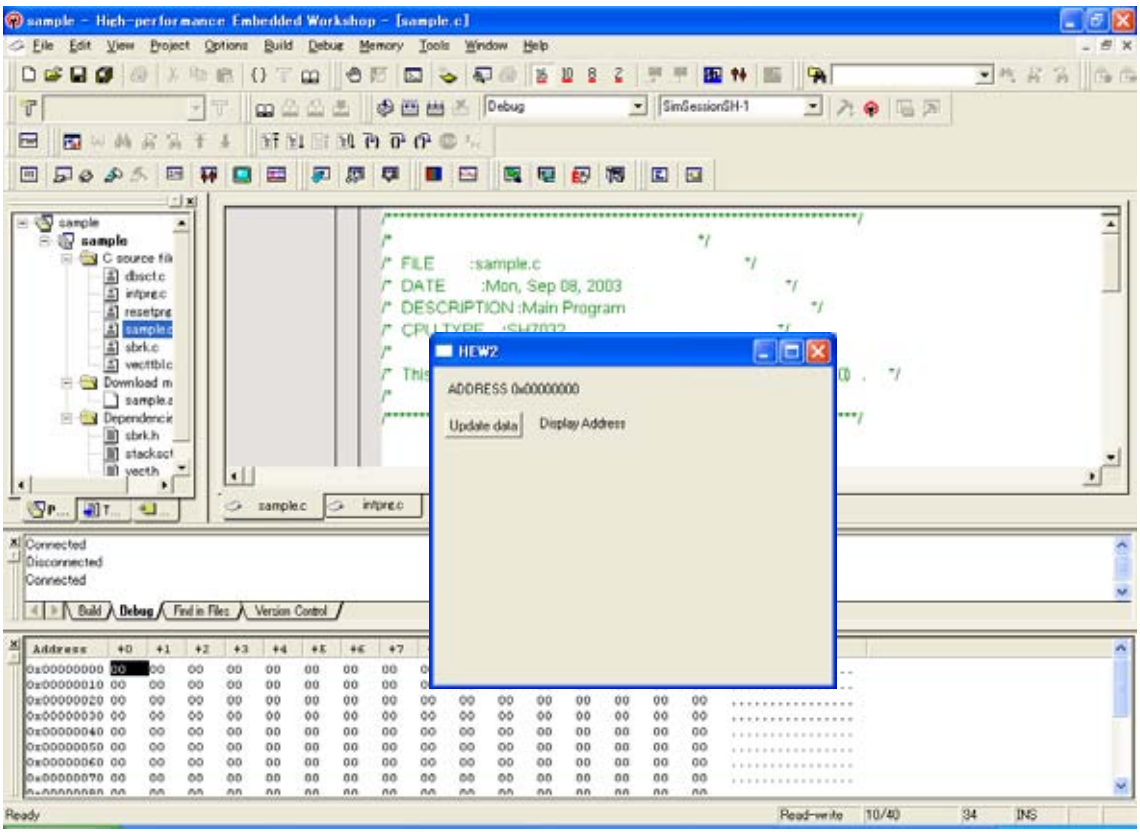


図　起動画面

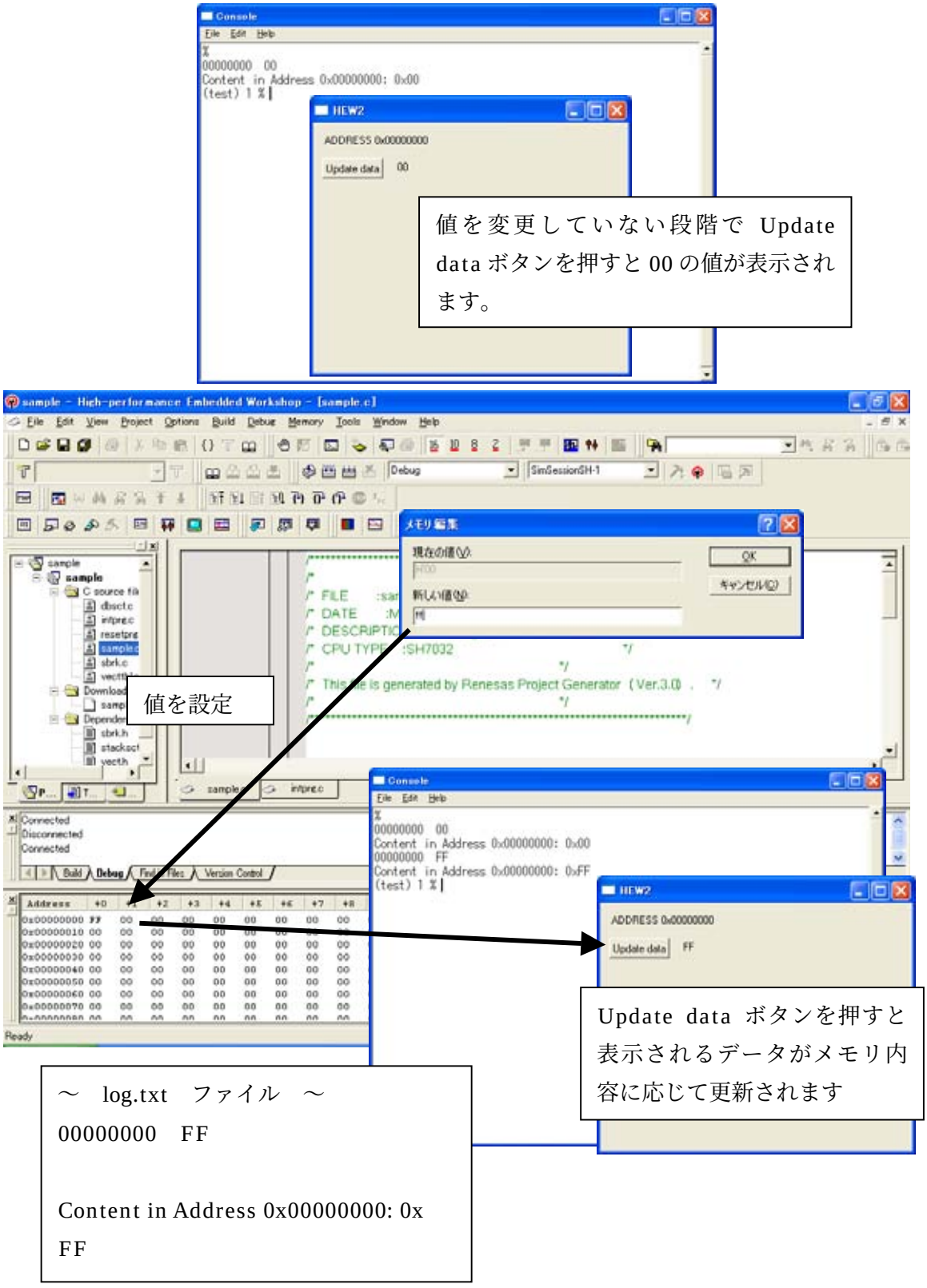

Console
File Edit Help
%
00000000 00
Content in Address 0x00000000: 0x00
(test) 1 %
HEW2
ADDRESS 0x00000000
Update data
00
値を変更していない段階で Update data ボタンを押すと 00 の値が表示されます。
sample - High-performance Embedded Workshop - [sample.c]
File Edit View Project Options Build Debug Memory Tools Window Help
Debug
SimSessionSH-1
メモリ編集
現在の値(V):
新しい値(N):
OK
キャンセル(C)
値を設定
Console
00000000 FF
Content in Address 0x00000000: 0xFF
HEW2
ADDRESS 0x00000000
Update data
FF
Update data ボタンを押すと表示されるデータがメモリ内容に応じて更新されます
～ log.txt ファイル ～
00000000 FF
Content in Address 0x00000000: 0x FF

図　実行例

HEW Tcl/Tk アプリケーションノート

発行年月日　2003年10月6日　　Rev.1.00

発行　　株式会社ルネサス テクノロジ　営業統括部
〒100-0004　東京都千代田区大手町2-6-2

編集　　株式会社 ルネサス ソリューションズ　ツール開発部

HEW Tcl/Tk
アプリケーションノート

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0932-0100