Palm Tcl: Programming Guide and Reference

Tcl Programming for Palm OS

DRAFT V0.4

© Ashok P. Nadkarni
E-mail: palmtcl@yahoo.com

Table of Contents

Introduction to Palm Tcl

This chapter gives a high level overview of the Palm Tcl language and development environment.

Palm Tcl is a high level programming language and development environment for creating applications for handheld devices running the Palm OS operating system. It allows programmers to write applications without requiring them to know the intricacies of the Palm OS runtime environment. Programs written in Palm Tcl are generally an order of magnitude shorter than their C equivalents and can be written in a fraction of the time.

About Palm OS

Palm OS is the operating system running on the most popular series of *personal digital assistant (PDA)* handheld computers – the *Palm Organizer* series from 3Com and Palm Computing, the *Visor* from Handspring and *Clie* from Sony.

Because it has been custom designed for a PDA environment, Palm OS has some features as well as hardware resource limitations that impose an additional burden on programmers who wish to write applications for this platform.

Palm Tcl's goal is to simplify and accelerate the process of application development for Palm OS. In its current version (0.3), Palm Tcl supports most form based user interface elements in Palm OS V3.0, as well as support for databases and several system related functions.

Applications written in Palm Tcl require at least V3.0 of Palm OS to run.

About Tcl

John Ousterhout at Berkeley created `Tcl`, the `Tool Command Language`, in 1988. It was designed as an open-source scripting language that was easily learned and yet powerful enough to be used in a wide variety of problem domains.

Even readers who never land up using Palm Tcl are strongly encouraged to learn `Tcl` and its associated graphics toolkit `Tk`. Few other development environment that comes close to its combination of benefits - ease of learning, speed of development, portability to a wide variety of platforms and applicability to a wide variety of application domains ranging from user interfaces to systems integration to distributed processing.

Why use Palm Tcl

Like Tcl, Palm Tcl allows newcomers to Palm OS to be functional in a day, instead of weeks or months. Even experienced programmers are far more productive.

When not to use Palm Tcl

Palm Tcl is interpreted, not compiled, and as such is not suitable for applications that are CPU intensive, for example those that require serious number crunching. Palm OS is not really a suitable platform for these applications anyway.

Futures and limitations

Palm Tcl is still under development and many Palm OS features are not yet supported. These include font and color support, networking and IrDA, and conduits. These will be added in a future release.

About this manual

This manual is a user guide and reference for the Palm Tcl development environment and language. It assumes that the reader is very familiar with the user interface provided by Palm OS and has programmed in Tcl on other platforms.

The manual uses the following typographical conventions:

> Palm Tcl program commands and resource definition file keywords that are to be entered verbatim are shown in this **`bold fixed size font`**.
>
> Parts of a Palm Tcl command or resource definition file that are placeholders for the actual characters are shown as upper case in this *`ITALIC FIXED SIZE FONT`*.
>
> Optional parts of a Palm Tcl command are shown surrounded by question marks as in **`?-var`** *`VARNAME`*?.

Other Reading

Reading about Palm OS

This manual contains some introductory material about the Palm OS environment. This is likely to be sufficient for simple programs but for more substantial applications, you will have to refer to the official Palm OS SDK documentation. The general concepts and features are described in **[5]** while the detailed reference is contained in**[6]**.

There are several books on Palm OS programming available, for example **[1,2,3]**. Naturally, these focus on writing applications in C. Nevertheless, they are useful for understanding the Palm OS environment and some of its features such as user interfaces, databases and conduits. Some of these are explained in better detail in the books than the official Palm OS SDK documentation.

Reading about `Tcl`

This manual contains no introductory or reference material about Tcl. It assumes the reader is familiar with the language and only describes how Palm Tcl differs. In addition to detailing the new commands, the differences with respect to the core language are described in Palm Tcl compatibility with Tcl 7.6.

For readers who are new to Tcl, there are a number of excellent books listed in the References section. Of these, **[4]** is the most relevant because it describes Tcl 7.6, the version on which Palm Tcl is based. There are several other references listed in http://starbase.neosoft.com/~claird/comp.lang.Tcl/Tcl_books.html. These describe a newer version of the language but most of the material is still relevant.

In addition to the books, there is a wealth of material available on the Web. The Scriptics Corporation web site at http://www.scriptics.com is the logical starting point for exploring `Tcl` resources on the Web. Another very useful Web page for the beginner is http://starbase.neosoft.com/~claird/comp.lang.Tcl/Tcl_tutorials.html which contains a directory of online `Tcl` tutorials. A detailed guide to beginning development with `Tcl` is the *Getting Started* section at http://dev.scriptics.com.

Credits

The Palm Tcl development environment is built from several free open source utilities.

The `Tcl` language

Obviously, without `Tcl`, there would be no Palm TCL. In addition to John Ousterhout who invented the language, there is big community of folks on comp.lang.tcl and elsewhere who keep the language moving forward. At the time of this writing, the version of Tcl is at 8.3 and is considerably more advanced than the version 7.6 on which Palm TCL is based. Unfortunately, the latest version has also grown too big to be suitable for a Palm-sized environment.

The `prc-tools` C development tools

Palm TCL is itself written in C using the prc-tools package. This is a port of the Gnu gcc compiler chain, by John Marshall and others, to support Palm OS applications. Without the availability of a free compiler, Palm TCL would never have come into being.

The `PilRC` resource compiler

Although any Palm OS resource compiler may be used, the one used in sample code and described in this manual is PilRC (PILot Resource Compiler). This is a freely available resource compiler that is downloadable from its official home on the Internet at http://www.ardiri.com/index.php?redir=palm&cat=pilrc . A version of this tool is also included in the Palm TCL distribution.

The `par` resource database utility

The `par` utility is used to bind the application's resources to the Palm Tcl runtime system. This program, written by David Williams, is a general-purpose tool for manipulating Palm OS resource databases and can be downloaded from http://djw.org/product/palm/par/index.html.

Toucan – the Palm TCL Integrated Development Environment

Toucan is an integrated development environment for Palm TCL written by Mac Cody. Amongst other things, it allows for graphical creation and viewing of form elements, making creation of the latter significantly simpler than editing resource files. Documentation and software for the latest version of Toucan can be downloaded from http://home.attbi.com/~maccody/.

Palm OS Features

Provides a short introduction to the Palm OS environment and the features supported by **Palm Tcl**.

Like other operating systems, Palm OS provides a fairly extensive programming interface covering user interfaces, storage, networking and communications. In addition, there are several aspects of Palm OS that are unique, such as conduits and categories. This chapter introduces you to those features of Palm OS that are supported by **Palm Tcl**. It is *not* meant to be anything other than a very rudimentary introduction but should be sufficient to get started. Refer to **[5]** and **[6]** for a detailed description of Palm OS.

User Interface Elements

Resources

A Palm OS application is stored as a *resource* database. A *resource* is a data structure that contains a component of an application, for example its code, or icon, or structures that describe its user interface elements such as forms, menus etc.

The user interface elements in a Palm OS application are generally not created dynamically. Rather, they are defined and created as static structures through a program such as `pilrc`. These programs take a *resource definition file* that describes the static structure of the user interface, such as position and size of the user interface elements, and converts it to a Palm OS resource. The application program code is then responsible for the dynamic aspects of the user interface, such as handling button taps, displaying dynamic content and so on.

All the user interface elements discussed in the rest of this section are defined and created through a resource definition file. Although any resource creation program may be used, the documentation assumes you are using the `pilrc` (PILot Resource Compiler) program. This is a free program that is available from the Internet from http://www.hig.se/~ardiri/development/palmIII/pilrc/index.html. The example **Palm Tcl** application Demo - resource definition file illustrates some of the basic features of the program. However, this manual does not document all the `pilrc` features or the syntax of the resource definition file it uses. Refer to the documentation that comes with that program for a full description.

Resources are identified by a *resource type code*, which indicates the type of resource (string, button, text field etc.) and a *resource id,* which is a number between 1 and 65535 that is used to distinguish between multiple resources of the same type. Resources are loaded and manipulated using these two items. Generally, the type of resource is implicit in the function being invoked, while the resource id is explicitly specified.

Note that all resource ids above `10000` are reserved by Palm OS for system use. Moreover, Palm Tcl reserves the use of the resource id `9999`. This resource id must be associated with a string resource that contains the Palm Tcl script to be invoked when the application starts.

Forms

Although Palm OS does provide low level windowing and graphics routines, Palm Tcl is focused on applications that are based on Palm OS *forms*[1]. Using terminology from other windowing systems, a Palm OS form is basically a container window with a title and menu that collects together various user interface elements, such as buttons and text fields, that are functionally related. An application usually consists of a number of these forms that the user navigates through using buttons or menu selections.

Form resources are defined using the **`FORM`** keyword in a `pilrc` resource definition file and appear at the syntactic top level. User interface elements contained within a form are enclosed within the definition of the parent form.

The following defines a form in a resource definition file.

```
FORM ID 500 AT (0 0 160 160)

MENUID 550
NOFRAME

BEGIN
  TITLE "Palm TCL Demonstration"
  FIELD ID 501 AT (10 20 140 80) FONT 0 NONEDITABLE MULTIPLELINES
  SCROLLBAR ID 502 AT (150 20 7 80) VALUE 0 MIN 0 MAX 0 PAGESIZE 0
  BUTTON "User Interface Demo" ID 503 AT (40 120 AUTO AUTO) FONT 0
  BUTTON "Address Book Demo" ID 504 AT (40 140 AUTO AUTO) FONT 0
END
```

Defines form position and

Resource id of the menu associated with the form

Indicates no border to be drawn around form

Start of form element defintions

End of form definition

In Palm Tcl, loading a form resource through the **`form load`** command instantiates a Tcl command which is used to access or manipulate the form and all the user interface elements contained within it. The following code loads and displays the above form

.

```
set form [form load 500 -menucmd {menuproc %W %S}

$form set 501 {This is a simple demo}
```

Loads a form

Function menuproc will be called when a menu item is selected while the form is active

Initializes a form element

---

[1] This is true of most Palm OS applications. Games are a notable exception.

```
$form itemconfig 503 -command {%W unload; loadnextform}

$form display
```

Binds a command to a button on the form to unload the form and call a function to display a different one

Displays the form

As shown in the sample, the normal sequence for displaying a form is

- Load the resource containing the form definition, optionally defining the Palm Tcl procedure to invoke for menus attached to the form

- Initialize the various form elements within the form, including binding commands to be executed when elements such as buttons are selected

- Display the form

Multiple forms may be loaded simultaneously. However, in order to conserve limited resources, it is recommended that forms be unloaded when not in use.

Menus

Menus are defined in a `pilrc` resource definition file through the **`MENU`** keyword. Unlike most other form elements, menu definitions appear at the top level in the file and not inside a **`FORM`** definition. A menu is associated with a form through the **`MENUID`** form attribute (see Forms). This takes a parameter that is the resource id of the menu to be attached to the form.

A single menu definition may be shared between multiple forms by specifying the same menu resource id as the **`MENUID`** parameter for all the forms.

Within a **`MENU`** definition, the **`PULLDOWN`** `pilrc` keyword may be used define submenus. Within each submenu, the **`MENUITEM`** keyword defines individual menu items that may be selected by the user. A sample menu definition is shown below:

```
MENU ID 550
BEGIN
  PULLDOWN "Demos"
  BEGIN
    MENUITEM "UI Demo" ID 551
    MENUITEM "Address Book Demo" ID 552
  END
END
```

Defines a menu resource

Defines a submenu pulldown

Define an individual menu

End submenu definition

End menu definition

In Palm Tcl, menu selections are handled by associating a script with a form through the **`-menucmd`** option to the **`form load`** command. This script is run whenever the user makes a menu selection while the associated form is active. The script may contain event keywords that are substituted before the script is executed as described in Event Keyword Substitution. Two event keywords - **`%s`** and **`%W`** - are particularly useful for menu events. The **`%s`** event keyword in the script is replaced with the resource id of the menu item being selected. The **`%W`** keyword is replaced with the active form handle to which the menu is attached. This is useful when a menu is shared among multiple forms.

The example in Forms showed how a callback function `menuproc` was registered through the **`–menucmd`** option. The code fragment below shows a possible implementation of this function:

Corresponds to %W event keyword specified in the –menucmd parameter.

Corresponds to %S event keyword specified in the –menucmd parameter and is the resource id of the selected menu item. The function switch statement executes code based on the menu item selected.

```
proc menu {form id} {
 switch -exact -- $id {

551 {$form unload; source 9997}
  552 {$form unload; source 9998}
  default { fatal "Unknown menu item $id" }
 }
}
```

Titles

The **`TITLE`** keyword may be used within a **`FORM`** definition in a resource definition file to define the string to be shown as the title of a displayed form. This string is statically defined through the resource file and Palm Tcl does not provide any means of accessing it. See the example in Forms for a sample definition.

Labels

A *label* resource is used to display noneditable text strings in a form. Labels are defined in a resource definition file through the **`LABEL`** keyword.

Palm Tcl does not provide any means of modifying a label and hence labels should not be used to display strings that may change at runtime. The value of a label resource may be obtained through the *`FORMHANDLE`* **`get`** command.

☛ Use a *text field* with the **`NONEDITABLE`** attribute to display noneditable text that can be dynamically changed at runtime. However, do not use text fields for static text that does not change since fields take up more memory resources than labels.

Text fields

A *field* is a user interface element that displays one or more lines of text. Fields are defined in the resource file through the **`FIELD`** keyword and have several attributes associated with them such as fonts and alignment. The `pilrc` keywords for some of the common attributes are shown in Table 1.

`FONT`	Defines the font number to be used for the field.
`LEFTALIGN/RIGHTALIGN`	Specifies the text alignment within the field.
`EDITABLE/NONEDITABLE`	Controls whether the user can edit the field or not.
`SINGLELINE/MULTIPLELINES`	Specifies whether the field can extend over multiple lines.
`UNDERLINED`	If specified, the whole field is underlined. This is usually set on editable fields as a visual cue to the user that the field can be modified.
`MAXCHARS`	The maximum number of characters that can be allowed in the field. **Because of a bug in some versions of `pilrc`, this attribute must be present for editable fields to work correctly.**
`HASSCROLLBAR`	This attribute does *not* display a scrollbar for the field. It is only used to indicate to Palm OS that additional events related to scrolling should be generated for this field. This attribute is not required with Palm Tcl even if the field needs a scroll bar.

Table 1

The contents of a field may be programmatically retrieved or set using the *`FORMHANDLE`* **`get`** and *`FORMHANDLE`* **`set`** commands respectively.

Field contents may exceed the display area available for the field. In this case, the field must be associated with a scroll bar that must be separately defined in the resource file and positioned appropriately next to the field. Although Palm OS itself does not automatically support a scroll bar for fields as it does for lists, it is easy to support the functionality in Palm Tcl by associating the scroll bar with the field through the *`FORMHANDLE`* **`itemconfig`** command. This command must be executed after the form is loaded but before it is displayed on the screen.

The following resource definition lines (assumed to be inside a **`FORM BEGIN..END`** block) define a field resource and a scrollbar that will be programmatically associated with the field.

Defines a field element that extends over multiple lines

```
FIELD ID 1401 AT (10 20 140 50) FONT 0 UNDERLINED MULTIPLELINES MAXCHARS 256
SCROLLBAR ID 1402 AT (PrevRight PrevTop 7 50) VALUE 1 MIN 1 MAX 1 PAGESIZE 1
```

Define a scrollbar. Nothing in the resource file associates this with the field

Position the scroll bar right next to the field. The height of the scrollbar should equal that of the field.

The following code fragment then attaches the scroll bar to the field. Palm Tcl then automatically handles all scrolling.

```
$form itemconfig 1402 -scroll 1401
```

Resource id of field to scroll when scrollbar is moved

Resource id of scrollbar

Lists

A *list* is used for displaying a set of choices to the user. It is defined in the resource file by the **`LIST`** keyword, which must appear within a **`FORM`** definition.

The text strings to be displayed as the choices in the list may be included as part of the **`LIST`** definition. This is preferable if the set of choices is static and does not change. Alternatively, the set of choices displayed can be dynamically changed through the *`FORMHANDLE`* **`set`** command and retrieved through the *`FORMHANDLE`* **`get`** command. Both these take a Tcl list as an argument.

Users can select a list item by tapping on it. This deselects the currently selected item if any, and highlights the tapped item. The selected item can be retrieved in Palm Tcl through the *`FORMHANDLE`* **`getsel`** command. Scripts can be registered for execution in response to a user selecting a list item. This is done through the **`-command`** option to the *`FORMHANDLE`* **`itemconfig`** command. The script may contain event keywords that are substituted before the script is executed as described in Event Keyword Substitution.

The following example illustrates some of these techniques. The resource definition lines (assumed to be inside a **`FORM BEGIN..END`** block) define a list and a field resource that will be display the currently selected item in the list. In this example, the list is populated programmatically. If the items in the list were fixed and known before hand, they could simply have been included in the resource definition.

Defines a list

List elements. In this example, we will programmatically fill the list so we only define a single element here

Indicate that 4 items should be visible at a time

```
LIST "Dummy" ID 1302 AT (10 10 70 AUTO) FONT 0 VISIBLEITEMS 4
  FIELD ID 1304 AT (10 PrevBottom+2 140 20) FONT 0 NONEDITABLE MAXCHARS 255
```

The following code populates the list and then displays the list item that the user clicks on in the field.

```
set items {}
```

Create a Tcl list in variable items

```
for {set i 0} {$i < 10} {incr i} {
  lappend items "This is item $i"
}
$form set 1302 $items

$form itemconfig 1302 -command "$form set 1304 \[$form getsel 1302\]"
```

Show the items in the displayed list

Binds a command to set the field to be executed when an item in the list is selected.

Gets the currently selected list item. Note the escaping of the [and] characters since we want to get the selection when the command executes, not when it is bound to the

Palm OS will automatically provide scrolling support for a list if required. No special programming needs to be done for this.

In addition to being directly displayed, lists may also be used in conjunction with a popup triggers to display popup lists that are visible only when the user taps on a control.

Scroll bars

Scroll bars are used in Palm OS in conjunction with fields, lists, and tables when the contents of the user interface element do not fit within the element's display area. Palm OS only supports vertical scroll bars, not horizontal. Scroll bars are defined in the resource definition file through the `SCROLLBAR` keyword.

Palm OS automatically handles scroll bars and scrolling for lists but not for fields or tables. It is easy in Palm Tcl to support scroll bars for fields through the command

```
FORMHANDLE itemconfig SCROLLRESID -scroll FIELDRESID
```

This takes care of all scrolling requirements for the field. See `FORMHANDLE itemconfig` for details. An example of using a scroll bar with a field is shown in Text fields.

The current state of the scroll bar (size, scroll position etc.) can be retrieved and set through the `FORMHANDLE get` and `FORMHANDLE set` commands. However, this is rarely required.

Buttons and repeating buttons

Buttons and repeating buttons display a text string in a (optional) box and can be tapped by the user to invoke a command. They are defined in the resource file through the `BUTTON` and `REPEATBUTTON` keywords respectively. Various attributes each keyword control the visual appearance of the button such as the presence of a surrounding box.

The difference between the two types of buttons is that a normal button sends a single event to the application in response to a user tap. A repeating button will keep sending events to the application until the user lifts the stylus from the button.

A Palm Tcl program can respond to both types of events by associating a script with the button. This is done through the **-command** option to the *FORMHANDLE* **itemconfig** command. The script may contain event keywords that are substituted before the script is executed as described in Event Keyword Substitution.

The example below illustrates use of a repeating button. The resource definition lines (assumed to be inside a **FORM BEGIN..END** block) define a repeating button and a field that will display a running count of events generated until the stylus is lifted from the button.

Defines repeating button resource

AUTO keyword indicates height and width of button is automatically calculated based

```
REPEATBUTTON "Repeat" ID 1702 AT (10 10 AUTO AUTO) FONT 0
FIELD ID 1703 AT (80 PREVTOP 30 30) FONT 0 NONEDITABLE UNDERLINED
```

The corresponding script registers a command to increment and display the value of a variable as long as the repeat button remains pressed.

Variable to hold count of button events

Initialize the field

```
global repeat_count
set repeat_count 0

$form set 1703 0
$form itemconfig 1702 -command {%W set 1703 [incr repeat_count]; %W redraw}
```

Callback command to increment counter and show it in the field on every button press

Force screen to be updated right away

Note the call to the *FORMHANDLE* **redraw** function. This is often required with a repeating button for the following reason – Palm Tcl redraws the display in the background when there are no events to be handled. This makes for more efficient processing allowing, for example, batching of screen updates. In the case of a repeating button, this can be confusing to the user. In the example above, if the explicit **redraw** command were not present, the variable `repeat_count` would be incremented continuously as long as the stylus remained on the repeating button. However, the screen would not be updated until the stylus was lifted at which point the final value of `repeat_count` would be displayed. The user would not see a continuous change in the value. Use of the **redraw** command forces the screen to be updated as the repeat_count variable changes. The user then sees the value changing incrementally as long as the button remains pressed.

Check boxes and push buttons

Check boxes and push buttons are both user interface elements that display two-state settings – on and off. They are toggled between the two states by user taps. Check boxes are on when they are checked and off when they are unchecked. Push buttons are on when they appear inverted and off when they have the default form foreground/background colors. Both types may have a text label associated with them.

Labels are shown next to a check box or inside a push button frame and can be changed through the `-label` option to the *FORMHANDLE* `itemconfig` command.

Check boxes and push buttons are defined in a resource file by the keywords `CHECKBOX` and `PUSHBUTTON` respectively. Their state can be read from Palm Tcl through the *FORMHANDLE* `get` command. Conversely, the state can be toggled by the *FORMHANDLE* `set` command.

You can also associate a Palm Tcl script to be executed when the user taps a check box or push button. This is done through the `-command` option to the *FORMHANDLE* `itemconfig` command. The script may contain event keywords that are substituted before the script is executed as described in Event Keyword Substitution.

Palm Tcl also supports grouping of multiple check boxes or push buttons in such a manner that *exactly* one in a group is on. This is similar to radio buttons in the Windows environment. Adding the `GROUP` attribute to a `CHECKBOX` or `PUSHBUTTON` definition accomplishes this grouping. The `GROUP` attribute has an associated parameter that is the *group number*. All check boxes or push buttons that belong to the same group should have the same non-zero group number. When a check box or push button in a group is turned on, either programmatically or by a user tap, the others in the group will automatically be turned off. Note that it is not possible (or logical) to turn *off* a check box or push button that belongs to a group. Attempting to do so will generate a Tcl error exception. You can only turn an element *off* by turning *on* some other element in the group.

Selector triggers

A selector trigger displays a label surrounded by a rectangular button frame. The size of the button changes when the text in the button is programmatically modified. These controls are usually used to display a popup dialog from which the user can make a selection. The selected value is then displayed as the selector trigger label. A common use of selector triggers is to display the Palm OS built in day and time selection dialogs.

A selector trigger is defined in a resource file by the keyword `SELECTORTRIGGER`.

You can also associate a Palm Tcl script to be executed when the user taps a selector trigger. This is done through the `-command` option to the *FORMHANDLE* `itemconfig` command. The script may contain event keywords that are substituted before the script is executed as described in Event Keyword Substitution. This command may do some computation or display a dialog and then set the label of the selector trigger through the `-label` option to the *FORMHANDLE* `itemconfig` command.

See Time and Date Selectors for an example.

Popup triggers

A popup trigger displays a text label to the right of a graphic icon that indicates that the element controls a popup list. Popup triggers and lists are used in combination. The popup trigger is defined in a resource file through the **`POPUPTRIGGER`** keyword and then associated with a list with **`POPUPLIST`** keyword that ties the resource id of the popup trigger with that of a list resource.

Usually, the label displayed in a popup trigger is automatically set based on the selection made in the popup list. However, it can also be set through the **`-label`** option to the *FORMHANDLE* **`itemconfig`** command.

The list associated with the popup trigger may be either defined in the resource file or dynamically created at run time. If dynamically created, the script should also set the label associated with the control appropriately. The list must have the **`NONUSABLE`** attribute else it will be visible event when the trigger is not tapped.

The following example shows a popup trigger that is associated with a dynamically created list.

List resource that will be associated with the popup

Note that the list must have NONUSABLE attribute

```
LIST "Dummy" ID 1902 AT (PrevLeft PrevBottom+2 70 AUTO) FONT 0 NONUSABLE
      VISIBLEITEMS 3
```

Define the popup trigger element

```
POPUPTRIGGER "Demo PopupTrigger" ID 1903 AT (7 82 AUTO AUTO) FONT 0

POPUPLIST 1903 1902
```

Associate the popup trigger (1903) with the list (1902)

The script itself does not have to do much except populate the dynamic list (see Lists) and set the initial popup trigger label.

Tables

Tables are used to display data in columnar format. Tables consist of cells, each of which is identified by its row and column position within the table. Although Palm OS supports table cells of many types, Palm TCL currently only supports table cells that are contain editable text and checkboxes.

A table is defined in a resource file by the keyword **`TABLE`**. The `pilrc` keywords for some of the common attributes are shown in Table 2.

`COLUMNS`	Specifies the number of columns in the table.
`COLUMNWIDTHS`	Specifies the width for each column type.

ROWS	Specifies the number of rows in the table.

Table 2

Tables are amongst the most complex user interface elements to program in Palm OS. Palm TCL hides much of this complexity by providing built in support for features such as scrolling. Nevertheless, tables still require significantly more programming than the other user interface elements.

Conceptually, a Palm TCL table can be thought of as a window into an array of data records. The position of this window in the set of records can be controlled either programmatically by the application or internally by Palm TCL in response to various user input events. Callback routines may be defined to return appropriate records to Palm TCL for display.

When a form containing a table is loaded, the table must first be initialized to set the type of each column. This is done through the **-coltypes** option to the *FORMHANDLE* **itemconfig** command. This must be done before the form is displayed and cannot be changed thereafter. Next, the table must be initialized with the number of elements in the record set and the index of the record that is the displayed in the topmost row of the table. This is accomplished through the **-num_items** and **-top** options to the *FORMHANDLE* **itemconfig** command. In the simple case where the size of the array is fixed and equal to the number of rows in the displayed table, these options need to be set only once. An example of this is when a table is used to display the fields of a single record in a database. In more complex tables, where the table displays part of the data array, the **-top** option may be used to display different portions of the data. Moreover, if new elements may get added or removed from the data, Palm TCL must be informed of the change through additional calls using the **-num_items** option.

In most cases, a table needs to be capable of scrolling. This is true even if the size of the data array is fixed and equal to the number of rows in the table. The reason for this is that if any columns contain editable text fields, the area occupied by the text field can grow and shrink causing other rows to be scrolled off the display (or scrolled in). To manage this without a lot of programming on the application's part, Palm TCL allows association of buttons with a table through the **-downscroll** and **-upscroll** options to the *FORMHANDLE* **itemconfig** command. Palm TCL then automatically handles the scrolling internally. Note that the effect of the buttons is to scroll tables a *page* at a time, as opposed to line by line scrolling. This is in keeping with the behavior of the built in Palm OS applications.

The contents of the table can be set and retrieved using the *FORMHANDLE* **tset** and *FORMHANDLE* **tget** commands. However, if scrolling is enabled in the table, then Palm TCL needs to have the ability to load the contents of any row on demand. This is done through by specifying a callback script through the **-datacmd** options to the *FORMHANDLE* **itemconfig** command. This callback is invoked whenever Palm TCL needs to retrieve data from the application to be displayed in the table or when it needs to notify the application to store the data after it has been modified.

Graffiti Shift Indicator

This section to be written.

Clipboard and Selections

Palm OS maintains a *clipboard* that can be used for cutting, copying and pasting of data between applications. Although Palm OS itself allows three types of data to be stored in the clipboard – text, bitmaps and ink – Palm Tcl only supports storing and retrieval of text data to and from the clipboard.

Access to the clipboard is provided through the `clipboard` command. Normally, this command should be used in a procedure that is bound to a menu item or keyboard shortcut to allow the user to copy selected data to and from the clipboard.

The clipboard is often used in conjunction with selections. Palm Tcl allows retrieval of the currently selected text with the *FORMHANDLE* **getsel** command. This command returns the currently selected text for field and list form elements. The sample code below shows a procedure that might be bound to a menu item to cut or copy text selected in a field element.

```
# Cut the current selection to the clipboard
# $form is the form handle
# $op is the operation - cut or copy
proc cutcopy {form op} {
  # Get the current focus and return if no focus
  set focus [$form focus]
  if ![string length $focus] return

  # We will only allow cut and paste for fields
  if [string compare [$form itemcget $focus -type] field] return

  # Get the selected range
  foreach {start end} [$form getsel $focus temp] break


  # Nothing selected so nothing to do
  if ![string length $start] return

  # Copy selection to clipboard
  clipboard $temp

  # All done if copy operation
  if [string compare $op cut] return

  # Delete selection from current field content
  set temp [$form get $focus]
  incr start -1
  set temp "[string range $temp 0 $start][string range $temp $end end]"
  $form set $focus $temp
}
```

$temp contains selection

$start and $end
contain offset of start
and end of selected

Conversely, the following code pastes the current clipboard contents into the field currently having focus.

```
# Paste the current selection to a field
proc paste {form} {
  # Get the current focus and return if no focus
  set focus [$form focus]
  if ![string length $focus] return

  # We will only allow cut and paste for fields
  if [string compare [$form itemcget $focus -type] field] return

  # Get current clipboard contents
  set ins [clipboard]
  set inslen [string length $ins]

  # If clipboard empty, nothing to paste
  if !$inslen return

  # Get current contents of field with focus
  set text [$form get $focus]

  # Find the current insert position …
  set inspos [$form itemcget $focus -inspos]
  # …and insert text there
  $form set $focus "[string range $text 0 [expr $inspos-1]]$ins[string range
        $text $inspos end]"

   # Finally set insert position to end of pasted text
  $form itemconfig $focus -inspos [expr $inspos+$inslen]
}
```

Handling user input

In general, applications do not need to be directly concerned with graffiti or user character input. The Palm OS widgets, such as fields, take care of handling input in most cases. Under some circumstances, applications may wish to capture input keys and somehow process these before the default handling of the Palm OS widgets. For example, the application may convert all input lower case characters to upper case. This can be accomplished by establishing an event handler for the form to trap key events. See Event handling for an example.

Users may also interact with the device through the various hardware and silkscreen buttons, such as the power button, the calendar button, the page-up button, the application launcher icon etc. Palm Tcl applications can capture and respond to these actions as well using the common event handling mechanism described in Event handling.

Palm Tcl also supports two utility features provided by Palm OS that are intended to help users with data input:

- The **`palm keyboard`** command lets users input characters using an on-screen keyboard instead of graffiti.

- The **`palm graffitihelp`** command displays a pop-up table showing the graffiti strokes for each character.

This section to be written.Graffiti functions

Keyboard popup

Time and Date Selectors

Palm OS provides dialog boxes that allow the user to select time and date ranges. These dialog boxes are flexible enough to support selection in a number of ways; for example, the date selector dialog box allows selection of a day, by day, week or month. Palm TCL provides access to these built-in dialogs through the `clock selectday` and `clock selecttime` commands.

An example of using time and date selectors in conjunction with a selector trigger is shown below. Tapping the selector trigger will display a date dialog that allows the user to select a specific date. The date is then displayed as the label of the trigger.

Defines a selector trigger with
an initial label Select Day

The currently displayed
label will control size of
trigger

```
SELECTORTRIGGER "Select Day" ID 1802 AT (10 10 AUTO AUTO) FONT 0
```

The script fragment below registers a command to pop up a date selector dialog when the trigger is tapped and displays the user-selected date in the trigger. The dialog returns either the binary time corresponding to the date selected by the user, or an empty string if the user cancelled the dialog.

```
$form itemconfig 1802 -label "Select a day" -command {
  set date [clock selectday]

  if {$date != ""} {%W itemconfig 1802 -label [clock format $date]}
}
```

Display date selection dialog

Change selector trigger label based
on date selected

Playing Sounds

Palm TCL provides access to some of the sound playing capabilities of Palm OS through the beep command. Only access to the basic system sounds is provided. There is no support for more sophisticated capabilities such as playing MIDI files.

Palm OS Events

Event handling

Palm OS generates several types of events corresponding to graffiti input, hardware button presses and other system events. The Event Reference describes the various types of events that are visible in Palm TCL, and how scripts may be invoked in response to them.

Palm Tcl allows an application to intercept all these various events through event handlers. In the current version of Palm Tcl, event-handling scripts are registered on a per form basis through the **`-command`** option to the **`form load`** command. It is then the responsibility of the script to de-multiplex all the events it is passed and handle the ones that are of interest. The example in Event generation illustrates this.

In addition to processing the events themselves, the event handling script may choose to either pass on the event to the default handlers or bypass them.

Note that unlike the mainstream version of Tcl/Tk, Palm Tcl does not allow more than one script to be registered (for a particular form). There is no chainining mechanism as such. Of course an application can always implement its own chaining framework through a single entry point if desired.

Event generation

In addition to handling events, it is also possible for a script to generate an event to simulate data entry or any other system event. This is accomplished through the event command. For example, the statement

```
event system power
```

will cause the system to power off just as if the power off button was pressed.

Event handling and event generation can be used in combination. The following example converts all input characters to a form to upper case. An event handling script is attached to the form when it is loaded. All events will be passed to this script while the form is active. The script checks to see if the event is a key event for a lower case character. If so, it generates a new event with the corresponding upper case character and returns a **`-break`** code so that the default handlers for the key event are not run. In other cases it returns normally to allow the default handlers to process the event.

```
set script {
  if [string compare %E key] return
  set ch “%A”
  set uch [string toupper $ch]
  if ![string compare $ch $uch] return
  event key $uch
  return -code break
```

Define a multi-line script to handle all events

Ignore all events that are not key events and let default handlers

Not lower case letter. Let default handlers process the

Generate a new event with the upper case character

Return a break code so that default handlers for the event are not invoked

```
}
```

end of script

```
set form [form load $form_resid -menucmd menucmd -command $script]
```

Register the script to be invoked to handle all events when the form is active

Data Manager

Storage of data in Palm OS does not[2] follow the traditional directory/file model generally used in the computing world. Rather, data in Palm OS is stored in *databases* that hold *records* whose format is application dependent. A Palm OS interface known as the Data Manager provides the interface to access and manipulate databases and records. Palm Tcl exposes a considerably simplified version of this interface at the expense of some efficiency and features. Some of the features not supported by this version of Palm Tcl are:

- categories
- secret/hidden records

These features will be added in future versions of Palm Tcl.

Data Manager Concepts

Database creator ID

All databases stored on a Palm OS device are associated with a *creator id*. Generally, the creator id for a database should be the same as the creator id of the application that created the database. When an application is deleted from a Palm OS device, all databases with the same creator id are also deleted. In Palm Tcl, the creator id for a database is specified at the time it is created through the `dm create` command.

Database type

Along with the creator id, each database also has a *type* field that may be used to distinguish between databases with the same creator id. Like the creator id, this must also be a four-character string. A common one to use for databases is simply the string **`data`**. Like the creator id, the type for a database is specified at the time it is created through the `dm create` command.

Database records

A database in Palm Tcl can be viewed as a variable size array whose elements (records) are identified and accessed based on their position. The term *record index* refers to the position of a record in the array. Each record is also associated with a unique id which can be used to track the record even its position (index) in the database changes during

---

[2] This is not strictly true. There are some streaming file interfaces but these are generally meant to help in porting applications and do not follow the Palm design philopsohy.

operations such as sorting. The *DBHANDLE* **recinfo** command allows mapping between record indices and their unique id's.

The value of a record in a Palm Tcl database is simply a Tcl list. An element of the list then becomes a field record. The record is created using the Tcl **list** command and individual fields are accessed using the **lindex** command.

Working with databases

Working with databases in Palm Tcl is fairly simple. Databases are created though the **dm create** command. The commands **dm list** and **dm exists** may be used to enumerate databases and to check for existence. The command **dm delete** may be used to delete an entire database.

A database has to be opened before it can be accessed. This is done through the **dm opendb** command, which returns a handle to the opened database. Similar to traditional file access interfaces, this command allows the caller to specify the mode - read-only, exclusive etc. - in which the database is to be opened.

Once open, the individual records in the database can be read and written through the *DBHANDLE* **get** and *DBHANDLE* **set** commands respectively (where *DBHANDLE* is the handle of the opened database). The following code snippet illustrates use of these commands:

```
# Create a database (firstname, lastname)
dm create testdb TEST data
set db [dm opendb testdb -rw]
$db set end [list John Ousterhout]
$db set end [list Larry Wall]
$db close

# Reopen the database for reading and iterate through the records
set db [dm opendb testdb -ro]
for {set i 0} {} {incr i} {
  set data [$db get $i]
  if {[llength $data] == 0} {
    # No more records
    break
  }
  # Do something with $data
}
$db close
```

In addition, if the **-var** *VARNAME* option is specified with the **dm opendb** command, database records may also be directly accessed using the standard TCL syntax for arrays. The array index must be an integer and corresponds to the index of the record in the database. As a special case, the string **end** may be used to append new records to the end of the database. The code snippet below is identical to the one above except that it uses this method of accessing the database.

```
# Create a database (firstname, lastname)
dm create testdb TEST data
set db [dm opendb testdb -rw -var names]
set names(end) [list John Ousterhout]
set names(end) [list Larry Wall]
$db close

# Reopen the database for reading and iterate through the records
set db [dm opendb testdb -ro -var names]
for {set i 0} {} {incr i} {
  set data $names($i)
  if {[llength $data] == 0} {
    # No more records
    break
  }
  # Do something with $data
}
$db close
```

Records may be deleted through the *DBHANDLE* **`delete`** command. Note that when a record is deleted, the indices of records that follow it in the database are all decremented by one.

Finally, the database is closed by calling the command *DBHANDLE* **`close`**. This releases any associated resources after closing the database. It is generally good practice to close databases explicitly even though any open databases are automatically closed when a Palm Tcl application exits.

Databases may be sorted using the *DBHANDLE* **`sort`** command. This command provides various options for the sort criteria including the sort algorithm, the record field to sort on, the sort order, and the record comparison type (integer, string etc.). The sort algorithm may be specified to be either the insertion sort or quicksort. The former should be used when adding or modifying a single record in a database that is already sorted. The latter should be used in all other cases including when changing the sort criteria for the database.

In many cases, an application may need to find the new position of a record (the current working record for example) after a sort. The following example shows a function to do that.

```
# Resorts the database and returns new index of record at position $recI
# dbH is a handle to the database
proc reposition {dbH recI} {
  # Get the unique id of the record at position $recI
  set id [$dbH recinfo getid $recI]
  # Resort using insertion sort assuming only one record is changed
  $dbH sort -method isort -index 0 -stricmp
  # Return position of record identified by the unique id
  $dbH recinfo getindex $id
}
```

System functions

Launching applications

The `palm launch` command invokes the Palm OS system application launcher. There is no mechanism in the current version of Palm Tcl to invoke any other application. Note that this also ends the current application and in essence is the equivalent of a tradition Tcl `exit` command.

Palm OS feature support

Not implemented

Conduits

This section to be written.

Palm Tcl Development Environment

Describes the tools and process for creating an application using Palm Tcl*.*

The Palm Tcl build process

The general process for building an application using Palm Tcl is as follows:

Step 1. Obtain a creator id for your application from Palm Computing (***trademark?)***

Step 2. Using any text editor, create a resource description file that defines the user interface elements. This step is no different than if you were programming Palm OS in C using the gcc based prc-tools development environment. Define a string resource in your resource definition file that points to your Palm Tcl program file(s). The resource id `9999` should refer to the script file that should be executed by the Palm Tcl engine at program startup. See Creating Palm OS resources.

Step 3. Write your Palm Tcl program. Your program need not be contained within a single file, although it is a little more convenient to structure it that way. Make sure the file that contains your startup code has a resource id `9999` in your resource definition file.

Step 4. Convert your resource definitions and Palm Tcl scripts into binary resource files using the `pilrc` program. This will create several *.bin files corresponding to your resources. See Creating Palm OS resources.

Step 5. Use the `par` program to link the user interface and Palm Tcl program binary resources created in *Step 3* with the Palm Tcl runtime engine. You also assign the program name, creator id and version number for the program in this step. The output of `par` is a Palm OS application `.PRC` file. See Linking resources with the Palm Tcl runtime.

Step 6. *(Optional)* Load the created application `.PRC` file into `pose`, which is a Palm OS emulator that runs on multiple platforms including Windows, Linux and the Mac. This allows you test your application much more conveniently than loading into a real device. See Testing using the Palm OS emulator.

Step 7. Load the application `.PRC` file into your Palm OS PDA using the Palm Desktop Install Tool or equivalent.

The tools and utilities required for these steps are described in this section.

Obtaining a Palm OS Creator ID

Applications for the Palm OS need to have a unique *creator id*. This is a four-character code that must be unique amongst all Palm OS applications as it is used internally by Palm OS for various purposes such as associating databases with specific applications. If you are creating an application for your own use, you can use a temporary creator id such as **`TEST`**. However, if you want to distribute the application, you should use an official creator id. You can do this by registering your four-character code at the Palm creator id registration page at http://www.palm.com/devzone/crid/cridsub.html.

Creating Palm OS resources

Creating Palm OS resources is a two-step process.

- First, a text file – the resource definition file – is created that describes the various user interface elements. This file has a specific syntax depending on the utility that will be used to process it.
- Second, a *resource compiler* processes the resource definition file and converts the textual description of the application resources into the binary resource format understood by Palm OS.

There are several tools available to create binary resources for the Palm OS and any of them should work with Palm Tcl. However, this manual assumes the use of `pilrc`.

Using this tool is very simple – it is invoked as

```
pilrc RESOURCEFILENAME
```

where *`RESOURCEFILENAME`* is the name of the resource description file. The program creates several binary files, each with a `.bin` extension, corresponding to each Palm OS resource defined in the resource description file. These files are used as part of the input to the link step as described in **Linking resources with the Palm Tcl runtime**.

There are a few things that should be included in the resource definition file in addition to the user interface resources:

- A resource describing the version of the application should be included through the `pilrc` **`VERSION`** keyword.

- A resource of type **`STRING`** should be defined for every file that contains a Palm Tcl script to be included in the application. Moreover, the script to be executed when the program starts up should be associated with the resource id **`9999`**.

The documentation available with the `pilrc` download provides all the details regarding syntax of the resource definition file and `pilrc` command line options. This manual does not duplicate that information but the sample resource file in **Demo - resource definition file** should provide a useful starting point.

Linking resources with the Palm Tcl runtime

Once the binary resources are created, they need to be linked to the Palm Tcl runtime system to create an executable Palm OS application, which is itself a resource database file consisting of code resources (the Palm Tcl runtime system), user interface resources (from the resource compiler output), and other resources like icon bitmaps and version strings. The Palm Tcl runtime system is itself shipped as a Palm OS application, `palmTcl.prc`, but does not have any resources contained in it except for the runtime code. There are several programs that may be used to combine the output of the resource compiler with the runtime system. The one described here is the `par` utility.

To bind the application resources with the Palm Tcl runtime system, the `par` utility is invoked as

```
par r PRCNAME.prc PROGRAMNAME appl CREATORID palmTcl.prc *.bin
```

A brief explanation of the command follows. More details on the various options available with `par` can be found in its documentation.

`r` is a function code to `par` that tells it to create a new resource database file (the application itself).

`PRCNAME` is the base name of the file to be created. The extension specified must always be **`.prc`**. The application is created as a file with name *`PRCNAME`***`.prc`**. This is the file that will be registered with the `Palm Desktop Install Tool` for loading into the PDA.

`PROGRAMNAME` is the name by which the application will be visible on the PDA.

`appl` indicates that the resource database being created is actually an executable application. This is the four-character database type stored with all resource databases in Palm OS.

`CREATORID` is the creator id you have registered with Palm Computing ***(trademark?)***

`palmTcl.prc` is the resource database shipped with the Palm Tcl development environment. It contains the runtime system for Palm Tcl. `par` will extract the code resources out of this and store them in *`PRCNAME`*`.prc`.

`*.bin` refers to the binary resource output files generated by the `pilrc` program. There is one file per resource and `pilrc` constructs each file name based on the resource type and its resource id. Instead of explicitly listing the files, it is usually easier to use the wildcard specification. However, care should be taken that there are no extraneous `.bin` files left over from previous versions or build attempts.

The file *`PRCNAME`***`.prc`** contains the application code and resources. It may be loaded into a Palm OS based device.

Testing using the Palm OS emulator

Before loading a Palm OS application into a real device, it may optionally be tested on the Palm OS emulator `POSE`. This application runs on multiple platforms including Windows and is available from http://www.palmos.com/dev/tech/tools/emulator. It includes an almost complete emulation of Palm OS. It is very useful for testing purposes as it can emulate several different Palm OS devices and also contains several features that aid in testing. Refer to the documentation that comes with it for details on how to load and test your application in this environment.

Palm TCL provides access to various emulator features through the emulator command. These are very useful during the development process, for example to load scripts during testing without having to rebuild the application. Note that these commands cannot be used when the application is running on a real Palm device.

Tracing using the Palm Reporter

Another very useful tool for debugging is `Palm Reporter`. This tool can be used to display trace messages while running your application inside `POSE`. The Palm TCL `log` command can be used to send execution trace messages to `Palm Reporter` so that you can track the execution of your program as it runs.

`The Palm Reporter` utility can be downloaded from the `POSE` home page at http://www.palmos.com/dev/tech/tools/emulator.

Loading an application into a Palm OS PDA

Once the application is built, it may be downloaded into the Palm OS device using the device's install utility.

For the Palm series of PDA's from 3Com or Palm Computing, this utility is the `Palm Install Tool` in the `Palm Desktop` program group in Windows. Start the application and use the **`Add`** button on the first dialog box to register your application for installation into your PDA. Run `hotsync` to synchronize your PDA and your application will be downloaded into the device.

Programming by Example

This chapter introduces the reader to the Palm Tcl development environment through an example.

This chapter introduces you to **Palm Tcl** through a simple example program – a rudimentary address book. It is assumed that the reader has knowledge of **Tcl** and only illustrates features that are specific to Palm OS.

Demo - resource definition file

Demo – program

Demo – compiling resources

Demo – linking

Palm Tcl Reference

This is the official reference for all Tcl commands that are specific to Palm OS.

This section details the Tcl commands that are specific to Palm Tcl and differences between the standard commands in Palm Tcl and those in the Windows/Unix version of Tcl.

Palm Tcl compatibility with Tcl 7.6

Palm Tcl is based on Tcl 7.6 but has some significant differences. These differences arise for several reasons:

Some commands have been removed because they are not relevant to the Palm OS environment. These commands are – **`cd exit file flush gets glob load open puts pwd read seek tell`**.

Dynamic memory and stack space is extremely limited in Palm OS devices. Some commands had to be removed because of their memory usage. Some (memory-optimized) form of these commands may be added in future versions. These commands are: **`history`**.

Some commands have not been implemented to reduce the Palm Tcl code footprint. These are: **`case interp package`**.

The following commands have not been implemented but will be in a future release: **`after close eof flush socket vwait`**.

The following commands have been partially implemented or modified for various reasons:

- The **`clock`** command does not have the **`scan`** subcommand but has been enhanced to add the **`selectday`** and **`selecttime`** subcommands.
- The **`source`** command has been modified to take a Palm OS resource id as a parameter instead of a file name. The corresponding resource should contain the script to be evaluated.
- The **`info`** command has additional options **`memstat`**, **`palmtcl_version`** and **`palmtcl_patchlevel`**

The following new commands have been added to support Palm OS features – **`clipboard dm form fatal log memstat`**.

Alphabetical Reference

beep

The **`beep`** command allows you to play one of the predefined sounds in Palm OS. It takes the following form:

`beep ?`*`BEEPTYPE`***`?`**	Plays the predefined sound indicated by *`BEEPTYPE`*.

Table 3

beep ?*BEEPTYPE*?

Plays one of several different system sounds. The argument `BEEPTYPE` indicates the sound to be played and must be one of the following: `info`, `warning`, `error`, `startup`, `alarm`, `confirm`, or `click`. If unspecified, the argument defaults to `alarm`.

clipboard

The clipboard command allows you to retrieve or store text from the Palm OS clipboard. It takes the following form:

`clipboard`	Retrieves contents of the clipboard.
`clipboard ?`*`text`***`?`**	Stores *`text`* into the clipboard.

Table 4

clipboard

Without any arguments, the **`clipboard`** command returns the current contents of the clipboard.

clipboard ?*text?*

If an argument is specified to the **`clipboard`** command, it is stored into the clipboard overwriting its current contents. An empty string is returned.

clock

The **`clock`** implements various time and date related functions. It takes one of the following forms:

`clock clicks ?-rate?`	Return system dependent integer value representing a high resolution timer.
`clock format` *`CLOCK VALUE`* **`?-format`** *`FORMAT STRING`***`?`** **`?-gmt`** *`BOOLEAN`***`?`**	Returns a formatted string corresponding to a binary time value.
`clock seconds`	Returns the number of seconds since the beginning of an unspecified epoch.

	unspecified epoch.
`clock selectday ?-by MODE? ?-init BINTIME? ?-title TITLE?`	Displays a dialog to the user to allow selection of a day
`clock selecttime`	Displays a dialog to the user to allow selection of a time range

Table 5

Note that the `clock scan` command available in TCL 7.6 is not implemented in Palm TCL.

clock clicks ?-rate?

Without the `-rate` option being specified, the command returns an integer value representing the number of clock *clicks* since some undefined instant. The returned values should be used for relative measurement of intervals. The actual interval corresponding to the each tick may be obtained by specifying the `-rate` option. In this case the command returns the number of clicks in one second. The `-rate` option is specific to Palm TCL and is not available in the standard TCL 7.6 distribution.

clock format *CLOCK_VALUE* ?-format *FORMAT_STRING*? ?–gmt *BOOLEAN*?

This command returns an ASCII representation of the binary time represented by *CLOCK_VALUE*. The binary time value is generally that returned by the `clock seconds` command. An optional formatting string may be specified to control the form and contents of the returned string. This command is unchanged from the standard TCL 7.6 distribution. Refer to the documentation of TCL 7.6 for details.

clock seconds

Returns the number of seconds since some undefined epoch. This can be used to measure intervals or passed to the `clock format` command for converting to human readable form.

clock selectday ?-by *MODE*? ?-init *BINTIME*? ?-title *TITLE*?

Displays a dialog that allows the user to select a specific date and returns the corresponding number of seconds since the epoch. Returns an empty string if the user cancels the dialog. The returned value can be passed to the `clock format` command for converting to human readable form.

The `-by` option is used to specify the mode in which the date selection is allowed: *MODE* may have the value `day`, `week` or `month` depending on whether the date selection is by day, week or month respectively. If the `-by` option is not specified, it defaults to `day`.

The `-init` option allows specification of a specific date to be displayed to the user as selected. The *BINTIME* value must be specified as the number of seconds since the epoch. It may have the special value `none` to indicate no date should be initially selected. If the `-init` option is not specified, it defaults to the current date.

The **`-title`** option allows specification of a title for the dialog box. This defaults to "Select a date" if the option is not specified.

clock selecttime ?–start *STARTTIME*? ?-end *ENDTIME*?-title *TITLE*?

Displays a dialog that allows the user to select a time range and returns a list containing the start and end of the selected time range. The time values are returned in units of seconds even though the dialog only allows selection on the granularity of a minute. An empty list is returned if the user cancels the dialog. If the user selects the **`No Time`** button in the dialog, a list of two empty elements is returned.

The **`-start`** and **`-end`** options allow specification of a default starting and ending time values to be displayed to the user. If unspecified, they default to the current time. Both *`STARTTIME`* and *`ENDTIME`* must be specified in units of seconds. As a convenience, the values specified for either are modulo a 24 hour clock so that binary date-time values (seconds since the epoch) may be passed in. This allows return value from **`clock seconds`** to be directly passed to this command.

The **`-title`** option allows specification of a title for the dialog box. This defaults to "Select a time" if the option is not specified.

dm

The **`dm`** command creates and manipulates Palm OS database resources. It takes one of the following forms:

`dm create` *`NAME CREATOR TYPE`* **`?-rsrc?`**	Creates a database of the given type, name and creator id.
`dm opendb` *`NAME ?options?`*	Opens the database with the given name and returns a handle to it. This is denoted by *`DBHANDLE`* below and is the name of a Tcl command that may be used to invoke operations on the database.
`dm exists` *`NAME`*	Checks whether a database of the given name exists.
`dm list ?-all? ?-name` *`NAME`*`?` **`?-type`** *`TYPE`*`?` **`?-creator`** *`CREATOR`*`?`	Returns a list of databases in the system with the given characteristics. *Not implemented yet.*
`dm delete` *`NAME`*	Deletes the database of the given name from the device.
`DBHANDLE` **`close`**	Closes the database referenced by DBHANDLE.
`DBHANDLE` **`get`** *`RECORDINDEX ?VARNAME?`*	Retrieves the value of a database record.
`DBHANDLE` **`set`** *`RECORDINDEX`* **`?-var?`** *`VALUE`*	Sets the value of a database record.

DBHANDLE **recinfo** *?options?*	Retrieves miscellaneous information associated with a database record.
DBHANDLE **count**	Returns number of records in the database.

Table 6

dm create *NAME CREATOR TYPE* ?-rsrc?

Creates a database with the given name and type. *NAME* is the name of the name to be associated with the database. CREATOR and TYPE specify the creator id and database type. The parameters to the command are shown in Table 7.

NAME	The name to be associated with the database.
CREATOR	The creator id to associate with the database.
TYPE	The type of the database.
-rsrc	*Not implemented yet.*

Table 7

If a database of that name already exists, the command raises a Tcl error exception. Callers should first check for the existence of the database using the **dm exists** *NAME* command.

The command returns an empty string.

dm opendb *NAME ?options?*

Opens the database with the given name and returns a handle to it. The database must exist and must not be already open else a Tcl error exception will be raised.

The options that may be specified with the command are shown in Table 8.

-ro	Specifies that the database should be opened in read-only mode. Any attempts to write to the database will result in a Tcl error exception.
-rw	Specifies that the database should be opened in read-write mode. Records may be read and written to the database. This is the default.
-exclusive	Specifies that the database should be locked. Attempts by other applications to open the database will fail until the database is closed.
-ignoresecret	If specified, when reading the database, records marked as secret are ignored when

	records marked as secret are ignored when iterating through the database.
`-var` *`VARNAME`*	Specifies a variable to be linked to the database. The database can then be accessed through this variable using array syntax.
`-fields`	*Not implemented yet.*

Table 8

The returned handle (denoted by `DBHANDLE` in the documentation) is also the name of a Tcl command that may be used to read and write the database. When the form is no longer required, it should be closed by invoking the command

```
DBHANDLE close
```

Deleting the corresponding Tcl command `DBHANDLE` will also have the same effect.

If the **`-var`** `VARNAME` option is specified, a global variable `VARNAME` is associated with the open database. The records of the database may then be directly accessed using the standard TCL syntax for arrays with the array index corresponding to the index of the record in the database. As a special case, the index **`end`** may be used to add a record to the end of the database.

dm exists *NAME*

Returns `1` if a database with the given name exists. Otherwise, returns 0.

dm list ?-all? ?-name *NAME*? **?-type** *TYPE*? **?-creator** *CREATOR*?

Not implemented yet.

dm delete *NAME*

Deletes the database with the given name. Note that once deleted, a database cannot be deleted using any of the Palm utilities.

DBHANDLE **close**

Closes the database associated with `DBHANDLE`. Returns an empty string.

DBHANDLE **get** *RECORDINDEX ?VARNAME?*

Retrieves the record at index `RECORDINDEX` in the database referenced by `DBHANDLE`.

If `VARNAME` is not specified, returns the record if found. If the record is deleted or `RECORDINDEX` is greater than the number of records in the database, returns an empty string. If an empty string is returned, caller should use the

```
DBHANDLE recinfo RECORDINDEX -state
```

command to check if the record is valid but empty, if it is deleted, or the record index in too large.

If *VARNAME* is specified, the command sets the variable of that name to the value of the record and returns `1`. Otherwise, the command returns `0` and sets the variable to be **`deleted`** or **`eof`** depending on whether the record has been deleted or *RECORDINDEX* is out of range.

DBHANDLE **set** *RECORDINDEX* **?-var?** *VALUE*

Sets the value of the record at index *RECORDINDEX* in the database referenced by *DBHANDLE*. If *RECORDINDEX* is greater than the number of records currently in the database or is the string **`end`**, a new record is appended to the database.

If **`-var`** is not specified, the database record is set to *VALUE*. If the **`-var`** option is specified, then *VALUE* is treated as the name of a variable in the current context that contains the actual value to use.

The command returns the index of the record modified or created. This is useful when appending a record to the end of the database. The returned index can be used to get the unique id of the record (through the **`DBHANDLE recinfo`** command) so that the record can be located after sorting the database.

DBHANDLE **delete** *RECORDINDEX?*

Deletes the record at index *RECORDINDEX* in the database referenced by *DBHANDLE*. In the current version of Palm Tcl, this is a complete removal from the database. It is not merely marked as deleted.

DBHANDLE **count**

Returns the number of records in the database.

DBHANDLE **sort** *?options?*

Sorts the records in the database based on the specified options. The possible options are shown in Table 9.

`-command` *TCLPROCNAME*	Specifies a TCL procedure to be invoked to compare two records. *Not implemented*
`-decreasing`	Records are sorted in decreasing order.
`-increasing`	Records are sorted in increasing order. This is the default.
`-index` *FIELDPOS*	Records are compared using the field at position *FIELDPOS* in the record. Defaults to 0
`-integer`	Specifies that records are to be compared based on integer comparison of the fields

	integer comparison of the fields
`-method` *`ALGORITHM`*	Defines the algorithm used for sorting. Valid values are **`qsort`** and **`isort`**
`-strcmp`	Specifies that records are to be compared based on case-sensitive string comparison of the fields
`-stricmp`	Specifies that records are to be compared based on case-insensitive string comparison of the fields. This is the default if **`–command`**, **`-strcmp`** and **`–integer`** are not specified.

Table 9 Database sorting options

When sorting the database, records are compared by either calling the procedure specified with the **`–command`** option or by directly comparing fields at position *`FIELDPOS`* indicated through the **`-index`** option. These two options are mutually exclusive.

The **`–command`** option allows specification of a TCL procedure that should be invoked to compare two records. The procedure is passed three arguments – the handle to the database, the first record (as a list) and the second record. The procedure should return **`–1`**, **`0`** or **`1`** depending on whether the first record should be considered smaller, equal or greater than the second record.

When the **`–index`** option is present, the **`–integer`**, **`–strcmp`** and **`–stricmp`** options may be used to control the mode of the comparison. By default, **`–stricmp`** is presumed if nont of these options is present.

The **`–increasing`** and **`–decreasing`** options specify the order in which records are to be sorted. If multiple such options are specified, the last one takes effect.

The sorting algorithm to be used can be specified through the **`–method`** option. Specifying **`qsort`** as the argument for the options causes the Quicksort algorithm to be used. Specifying **`isort`** as the argument causes the Insertion sort algorithm to be used. The insertion sort algorithm is quicker when the database is mostly sorted in the correct order, for example when a new record is added to an existing sorted database. In all other cases, the Quicksort algorithm is considerably faster and should be used. Specifying multiple **`–method`** options causes the last one to take effect.

DBHANDLE **recinfo** *?option?*

This function retrieves various information about a record. The various options are shown in Table 10.

`getid` *`RECORDINDEX`*	Returns a unique value that identifies the record at position *`RECINDEX`* in the database. This can be used with the **`getindex`** option to locate the record after

	resorting.
`getindex` *`RECORDID`*	Returns the position (index) of the record whose unique id is *`RECORDID`*.

Table 10 *DBHANDLE* **recinfo** command options

emulator

The **`emulator`** allows access to various Palm OS Emulator features. This is very useful during development, for example, to edit and modify scripts without having to rebuild the application and install it.

`emulator`	Returns 1 if the program is running in the Palm OS emulator, 0 other wise.
`emulator log` *`MESSAGE_STRING`*	Writes *`MESSAGE_STRING`* to the emulator's tracing facility.
`emulator source` *`FILENAME`*	Reads a file from the host system and does an `eval` on the contents.

Table 11 **emulator** command options

emulator

Without any additional arguments, this command returns 1 if the application is running under the `POSE` and 0 if running on a real device. The command should be used before using any of the other emulator subcommands.

emulator log *MESSAGE_STRING*

This command writes trace messages to the `Palm Reporter` tracing utility. It is only effective when running the application under `POSE`.

The command always returns the empty string.

emulator source *FILENAME*

This command reads a file from the host system that the emulator is running on and does an `eval` on the contents. Due to the memory constraints on the Palm OS, the file specified must not be more than 16000 bytes in size.

The primary purpose of the command is to be able to dynamically load script files from the host system during the development/test process. This command behaves similar to the `source` command in the standard TCL 7.6 distribution. Refer to the documentation of TCL 7.6 for details.

event

The **`event`** command allows programmatic generation of events defined in Event Reference. It may take one of the following forms:

`event key` *`KEYCODE`*	Inserts an event of type **`key`** into the event queue
`event menu` *`MENUID`*	Inserts an event of type **`menu`** into the event queue
`event system` *`SYSTEMCODE`*	Inserts an event of type **`system`** into the event queue

Table 12

event key *KEYCODE*

Inserts an event of type **`key`** into the event queue. The effect is as if the user had entered the *`KEYCODE`* ASCII character.

event menu *MENUID*

Inserts an event of type **`menu`** into the event queue. The effect is as if the user had selected the menu item corresponding to resource id *`MENUID`*.

event system *SYSTEMCODE*

Inserts an event of type **`system`** into the event queue. *`SYSTEMCODE`* must be one of the strings listed in Table 26 Event strings for **system** events.

form

The **`form`** command creates and manipulates Palm OS form resources. It may take one of the following forms[3]

`form ids`	Lists the resource ids of all form resources currently loaded into the application
`form handles`	Lists the handles of all loaded form resources
`form load` *`RESID`* **`?options?`**	Loads the form whose resource id is `RESID` into the application and returns a handle (`FORMHANDLE`) to it which is a Tcl command that may be used to manipulate the form.
`FORMHANDLE unload`	Closes the form if visible, and unloads it from memory.
`FORMHANDLE configure` **`?`***`options`***`?`**	Configures various options associated with an loaded form.

---

[3] No pun intended.

`FORMHANDLE cget ?options?`	Retrieves various configuration options associated with an loaded form.
`FORMHANDLE display ?-popup BOOLEAN?`	Makes the form visible if it is not already so and raises it above other forms that might be open.
`FORMHANDLE get RESID ?VARNAME?`	Retrieves the value of the form element whose resource id is `RESID`.
`FORMHANDLE set RESID ?-var? VALUE`	Sets the value of the form element whose resource id is `RESID`.
`FORMHANDLE getsel RESID ?VARNAME?`	Retrieves the current selection in the form element whose resource id is `RESID`.
`FORMHANDLE setsel RESID STARTPOS ENDPOS`	Sets the selection in the form element with resource id `RESID` to the character range whose positions are given by `STARTPOS` and `ENDPOS`.
`FORMHANDLE focus ?RESID?`	Retrieves or sets the current focus in a form.
`FORMHANDLE redraw ?-force?`	Redraws all or part of a displayed form.
`FORMHANDLE itemconfig RESID ?options?`	Sets various options for a form element. The options are dependent on the type of the form element.
`FORMHANDLE itemcget RESID ?options?`	Retrieves the configuration options for a form element.

Table 13

form ids

Returns a list of the resource ids of all forms currently loaded into the application.

form handles

This command returns a list of handles to currently loaded forms. The handles are those returned by the **`form load`** command when the forms are loaded. The handles are returned in an arbitrary order.

form load *RESID ?options?*

Loads a form with the resource id *`RESID`* into the application and returns a handle to it.

The *`?options?`* argument for the command may include the following options:

`-command` *`TCLSCRIPT script`*	Specifies a script to be invoked for every event generated while the form is active.
`-menucmd` *`TCLSCRIPT script`*	Specifies a script to be evaluated when a menu item is selected.

`script`

Table 14

The form must not be already loaded; otherwise, a Tcl error exception is raised.

The returned handle (denoted by *`FORMHANDLE`* in the documentation) is the name of a Tcl command that may be used to manipulate the form and the user interface elements contained in it. When the form is no longer required, it should be removed from memory by invoking the *`FORMHANDLE`* **`unload`** command. Deleting the Tcl command corresponding to the form will also have the same effect.

If the **`-command`** option is specified, *`script`* is evaluated for all events whenever the form is active. This evaluation is done in the global context after undergoing the keyword substitutions defined in Event Keyword Substitution.

The **`-menucmd`** option is similar except that *`script`* is evaluated only when the user selects an item from the menu associated with the form. The most useful keyword substitutions in this context are **`%S`**, which gets substituted with the resource id of the selected menu item, and **`%W`**, which gets substituted with the handle of the active form.

FORMHANDLE unload

Closes the form referenced by *`FORMHANDLE`*, removing it from the display and unloading it from memory. If this is the topmost form on the display, the next form in the stacking order is redrawn.

This command also deletes the Tcl command referred to by *`FORMHANDLE`*, which become invalid after this command returns and should not be referenced.

`FORMHANDLE` **`unload`** always returns an empty string.

FORMHANDLE configure *?options?*

Not implemented.

FORMHANDLE cget *?options?*

Retrieves the value of an attribute or configuration parameter associated with a loaded form. The keywords that may be specified in *`?options?`* and the corresponding values returned are:

`-resid`	The resource id of the form referenced by *`FORMHANDLE.`*
`-geometry`	*Not implemented*

Table 15

FORMHANDLE display ?-popup *BOOLEAN*?

Displays the form referenced by *`FORMHANDLE`* and raises it to the top of the screen. The form is also made active and receives all input including menu events. This command must be executed after a form is loaded to make it is visible on the screen.

The **`–popup`** takes a Boolean argument that indicates whether the currently active form should be closed before this form is displayed. If the *`BOOLEAN`* argument is **`0`**, **`false`**, or **`no`**, the topmost form is closed before this form is displayed. This is the default when the **`-popup`** option is not specified. If the *`BOOLEAN`* argument is **`1`**, **`true`**, or **`yes`**, this form is overlaid on top of the currently active form, which is left open but made inactive.

The command always returns the empty string.

FORMHANDLE get *RESID ?VARNAME?*

Retrieves the current value stored in the form element whose resource id is *`RESID`*. If *`?VARNAME?`* is specified, the value is also stored in a variable of that name in the current context. The variable is created if it does not already exist.

The value returned depends on the type of form element as shown in the table below. Note that not all element types support the get operation. Invoking the operation on any element whose type is not listed in the table will generate a Tcl error exception.

Check box	**`1`** if the pushbutton is currently selected, **`0`** otherwise
Fields	The current contents of the field as a string.
Labels	The currently displayed string for the label.
Lists	A Tcl formatted list containing the list items
Push button	**`1`** if the pushbutton is currently selected, **`0`** otherwise
Scroll bar	A list of 4 integers corresponding to the current position of the scroll bar, the minimum value, the maximum value and the scrollbar height. ***Needs elaboration***.
Title	The currently displayed string as the form title

Table 16

FORMHANDLE set *RESID* ?-var? *VALUE*

Sets the value of the form element whose resource id is *`RESID`* to *`VALUE`*. If the **`–var`** option is specified, then *`VALUE`* is treated as the name of a variable in the current context that contains the actual value to use.

The command always returns an empty string.

The allowed values for the various form element types are shown in the table below. Note that form element types not shown in the table do not support this operation. Invoking it with the wrong value type or on any form element type not listed below will result in a Tcl error exception.

Check box	Any Tcl Boolean value – **`true`**, **`yes`** and **`1`** will show the checkbox as being ticked and **`false`**, **`no`** and **`0`** will show it clear.
Field	Any string that is shorter than the maximum field length defined for the field through the *`MAXCHARS`* attribute in the resource definition file.
Label	Any string that is shorter than the label defined in the resource definition file. *Not implemented yet. Currently raises a Tcl exception.*
List	A Tcl formatted list containing the list items to be shown in the list box.
Push button	Any Tcl Boolean value – **`true`**, **`yes`** and **`1`** will show the pushbutton as being selected and **`false`**, **`no`** and **`0`** will show it clear.
Scroll bar	A list of 4 integers corresponding to the current position of the scrollbar, the minimum value, the maximum value and the scrollbar height. This is the same format as the value returned by the *`FORMHANDLE`* **`get`** *`RESID`* command.
Title	Any string. *Not yet implemented. Currently raises a Tcl exception.*

Table 17

FORMHANDLE getsel *RESID ?VARNAME?*

Retrieves the current selection in the form element with resource id *`RESID`*.

If *`VARNAME`* is not specified, returns the selection text. If there is no selection in the form element or if the form element does not support selection, the command returns an empty string.

If *`VARNAME`* is specified, and the form element has selected text, the selection is stored in a variable with name *`VARNAME`* in the current context. The variable will be created if it does not exist. For form fields that are lists, the command returns the starting index of the selection. For form elements that are fields, the returned value is a list the first element of which is the offset of the first character of the selection in the field value and the second element is the offset of the character following the last character in the selection. For lists, the returned value is the index of the selected item in the list. For both form element types, if nothing is selected, the command returns an empty string and *`VARNAME`* is not affected (and will not be created if it does not exist).

FORMHANDLE* setsel *RESID STARTPOS ENDPOS

Not yet implemented.

FORMHANDLE* focus *?RESID?

Gets or sets the current focus in the form.

If *`RESID`* is not specified, returns the resource id of the form element that has the focus. If no form element has the focus, the string **`none`** is returned.

If *`RESID`* is specified, it must be the resource id of the form element that is to receive the focus. As a special case, *`RESID`* may be specified as **`none`** to remove the focus from all form elements. The command returns **`1`** if the focus change was successful, and **`0`** otherwise, for example, if the form element is not of a type that can receive the focus.

FORMHANDLE* redraw *?*-force*?

Redraws a displayed form. Normally, the display update is optimized to batch changes and to occur when the system is idle. In some cases, for example, when repeating buttons are used, it may necessary to update the display right away using this command.

If the **`-force`** option is not specified, the command will only redraw those form elements whose data has been modified. If the **`-force`** option is specified, the entire form will be redrawn.

FORMHANDLE* itemconfig *RESID ?options?

Sets configuration options and attributes of the form element with resource id RESID. The allowed options and values, if any, for each form element type are shown below. Any attempt to set an option that is not supported or has a bad value will result in a Tcl error.

The command always returns an empty string.

`-colhide`	Table	Hides a column in the table
`-colshow`	Table	Makes a table column visible
`-coltypes`	Table	Defines the column types for a table
`-command`	Button, check box, list, push button, selector trigger, repeating button,table	Script to be evaluated when the form element is selected
`-datacmd`	Table	Script to be invoked as a callback to save or load data into a table row.
`-dispcmd`	List	Script to call to retrieve the text to draw for an owner draw list box. *Not implemented.*

`-downscroll`	Button, repeating button	Resource id of a form element of type table that is to be scrolled down when this button is pressed
`-fixsize`	Table	Makes all rows in the table fixed height so that rows will not expand in height as data is entered
`-label`	Button, check box, push button, selector trigger, popup trigger, repeating button	Label to be displayed for the form element
`-num_items`	Table	Total number of items in table
`-rowhide`	Table	Hides a tablerow *Not implemented*
`-rowshow`	Table	Makes a table row visible *Not implemented*
`-scroll`	Scrollbar	Resource id of a form element of type field that is to be associated with the scroll bar
`-top`	Table	The logical table index associated with the topmost row of the table
`-upscroll`	Button, repeating button	Resource id of a form element of type table that is to be scrolled up when this button is pressed
`-usable`	All	Hides or shows the form element so that it is no longer visible on the screen. *Not implemented.*

Table 18

The **`-command`** option takes an argument that is a script to evaluated when the system generates an event indicating that the corresponding form element is selected. For buttons, push buttons and popup triggers, this happens when the form element is tapped. For lists, selecting a list item triggers the event is generated when the user selects a list item. For check boxes, the event is generated when the user checks or clears the check box. The associated script is evaluated in the global context after undergoing the keyword substitutions defined in **Event Keyword Substitution**. The most useful keyword substitutions in this context are **`%s`**, which gets substituted with the resource id of the form element, and **`%W`**, which gets substituted with the handle of the active form.

The **`-datacmd`** option takes an argument that is a script to evaluated to either retrieve or save data in a table row. Before being invoked, four additional arguments are appended to the script. The first argument is a function code. The second argument is the command associated with the form. The third appended argument is the resource id of the table itself. The fourth argument is the record number. Note that this option is mutually exclusive with the **`-dbattach`** option.

The **`-dbattach`** option (*not implemented*) allows direct table-based display and editing of a Palm Tcl database with no additional programming required. The option takes an argument that is a list, the first element of which is the name of the database, the second element is either **`-ro`** or **`-rw`** to indicate the database should be treated as readonly or read-write The remaining elements of the list correspond to column in the table and indicate which fields of a database record should be displayed in the column. Note that this option is mutually exclusive with the **`-datacmd`** option.

The function codes appended as the first argument to the script are shown in Table 19

`get`	The script should return a list each element of which corresponds to a column in the table
`save`	The indicated cell has been modified. The column number of the cell is passed as an additional argument

Table 19

The **`-scroll`** option is only applicable to scroll bar form elements. The associated argument must be the resource id of a field that is to be controlled through the scrollbar. Configuring a scrollbar using this option lets Palm Tcl take care of the machinery required to keep a field and an associated scrollbar updated when either one is changed.

The **`-upscroll`** and **`-downscroll`** options are only applicable to button and repeating button form elements. The associated argument must be the resource id of a table that is to be controlled through the button. Configuring a pair of buttons with these options lets Palm Tcl take care of the machinery required to keep a table and the associated buttons updated when either one is changed.

The **`-label`** option is applicable to button, check box, push button, selector trigger, and repeating button elements. The associated argument is the text to be displayed as the label for the element.

FORMHANDLE itemcget RESID ?options?

Retrieves the configuration options associated with a particular form element. Only the options shown in Table 20 are valid.

`-inspos`	Field	Returns the current insert position in the field.
`-num_items`	Table	Total number of items in table
`-top`	Table	The logical table index associated with the topmost row of the table
`-type`	All form elements	Returns the type of the form element which may be one of **`field`**, **`list`**, **`label`**, **`title`**, **`scroll`**, **`check`**, **`push`**, **`table`**, **`button`**, **`repbutton`**, **`seltrigger`**, **`poptrigger`**, **`poplist`** corresponding to the various element types.

Table 20 *FORMHANDLE* itemcget options

FORMHANDLE tget RESID ?*RECINDEX COLUMN*? …

Gets the values of cells in the table whose resource id is *`RESID`*. The *`RECINDEX COLUMN`* pairs indicate the locations of the cells whose values are to be retrieved. These are returned as a single list. The type of each element in the list is dependent on the type of that cell. Text cell types return a string value corresponding to the contents of the cell. Cells of type checkbox return 1 or 0 depending on whether the checkbox is selected or not.

FORMHANDLE tset RESID ?*RECINDEX COLUMN VALUE*? …

Sets the value of cells in the table whose resource id is *`RESID`*. The type and valid values of *`VALUE`* is dependent on the type of the corresponding table cell. For text cells, any string may be specified. For cells of type checkbox, the specified value must be a Boolean. Note that multiple *`RECINDEX COLUMN VALUE`* triples may be specified to set the values of multiple cells.

info

The Tcl command info, which returns various runtime information has been enhanced to take the additional options shown in

`memstat`	Returns a string showing memory usage statistics
`palmtcl_version`	Returns the version of Palm Tcl
`palmtcl_patchlevel`	Returns the version of Palm Tcl including patch information

Table 21

palm

The **`palm`** command provides access to miscellaneous Palm OS system routines.

`palm launch`	Invokes the Palm OS application launcher
`palm keyboard`	Displays a popup keyboard
`palm graffitihelp`	Displays help screens for graffiti strokes

Table 22

palm launch

This command exits the application and invokes the system application launcher.

palm keyboard

This command displays the Palm OS popup keyboard. The user can tap the displayed keys to enter data instead of using. Following standard Palm OS convention, this command should normally be bound to an item in the Edit menu.

palm graffitihelp

This command brings up the standard Palm OS help screens for graffiti input. This command should be bound to an item in the Edit menu.

source

The **source** command evaluates a script stored in a Palm OS resource. The command behaves similar to the **source** command in the standard TCL 7.6 distribution except that the script to be run is identified by a Palm resource id instead of a file name.

source ?*RES_ID*?	Reads a script stored in the Palm OS resource identified by *RES_ID* and evaluates it.

Table 23

Event Reference

Event Keyword Substitution

Palm TCL allows an application to register callback scripts that are invoked for specific events such as graffiti (character) input, selection events and hardware buttons. There are four event types:

key	Events of this type are generated when printable (generally ASCII) characters are entered through graffiti input or the keyboard display
system	Events of this type are used for system events such as pressing one of the buttons on the Palm device as well as virtual command keys entered using graffiti.
select	This type of event is generated when the user selects certain types of form elements like check boxes.

	elements like check boxes.
`menu`	Events of this type are generated when the user makes a menu selection.

Table 24

If callback scripts are associated with a form or form element, they are invoked after undergoing certain substitutions for `%` placeholders that are substituted with information specific to the event that caused the script to be invoked. These placeholders are replaced with the substitutions shown in Table 25 Event % substitution. Note that placeholders that have no valid substitutions for a particular event are replaced with the `??` sequence.

	`key` events	**`system`** events	**`select`** events	**`menu`** events
`%A`	Ascii character	See Table 26	`??`	`??`
`%E`	key	system	select	menu
`%N`	Integer corresponding to input character	Integer corresponding to input command character	`??`	`??`
`%R`	Resource id of form or `??` if no form active			
`%S`	Resource id of form element that currently has the focus		Resource id of form element being selected	Resource id of menu item being selected
`%T`	An integer corresponding to underlying Palm OS event type			
`%W`	The TCL handle for the form or `??` if no form active. The TCL handle is also the TCL command for invoking operations on the form.			
`%x`	??	??	Only valid for tables. Replaced with the column number of the selected cell.	??
`%y`	??	??	Only valid for tables. Replaced with the row number of the selected cell.	??

Table 25 Event % substitution

In the case of events of type system, the actual event is described by one of the strings shown in Table 26 Event strings for **system** events

`alarm`	Alarm is about to be generated
`autoOff`	Inactivity timer has triggered
`battery`	Battery is running low
`calc`	User selected the Calc silk screened button on device
`find`	User selected the Find silk screened button on device
`hard1`	User pressed the first hardware button on device (with the Date Book icon)
`hard2`	User pressed the second hardware button on device (with the Address Book icon)
`hard3`	User pressed the third hardware button on device (with the To Do List icon)
`hard4`	User pressed the fourth hardware button on device (with the Memo icon)
`hotsync`	User invoked Hotsync function through the device cradle
`launch`	User selected the Applications silk screened button on device
`left`	Move left graffiti character
`menu`	User selected the `Menu` silk screened button on device
`next`	Next field graffiti character
`pageDn`	User pressed page down hardware button
`pageUp`	User pressed page up hardware button
`power`	Power button has been pressed
`prev`	Previous field graffiti character
`right`	Move right graffiti character

Table 26 Event strings for **system** events

Appendix A - Version History

Describes the changes in each version of Palm TCL

Version 0.1

- Initial release

Version 0.2

- Added the `memstat` command
- Added emulator command
- Added `clipboard` command
- Added `clock` command including Palm TCL specific features
- Added **`-label`** option to the `FORMHANDLE itemconfig` command
- Added the `FORMHANDLE redraw` command
- Added support for repeating buttons, popup triggers and selector triggers
- Added dm exists command
- Added palm launch, palm keyboard and palm graffitihelp commands
- Added **`-command`** option to form load
- Added hooking for additional events such as hardware buttons and graffiti shortcuts
- Format for storing records in databases has changed. ***Incompatibility with version 0.1 databases***
- Added **`regexp`** and **`regsub`** commands (identical to TCL 7.6)
- Added the event command for generating events
- Binaries are built using Palm OS SDK 4.0 and `prc-tools 2.1pre3`

Version 0.3

- Added support for basic tables with scrolling

- Integrated apnlib module into and removed dependency on patched Palm OS header files

- Added **`memstat`**, **`palmtcl_version`** and **`palmtcl_patchlevel`** options to the info command

- Removed the **`memstat`** command (now part of the info)

Version 0.4

- Added **`-fixsize`** option to `FORMHANDLE itemconfig` command to make table rows non-expandable

- Added **`-var`** option to **`dm opendb`** command for accessing databases using array syntax

- Added **`-command`** option for table form elements

- Added the **`-inspos`** option to *`FORMHANDLE`* **`itemconfig`** and *`FORMHANDLE`* **`itemcget`** commands to set and get the insertion position for field elements

- Added the **`FORMHANDLE itemcget`** command option **`-type`** to retrieve the type of a form element

- Added the **`FORMHANDLE itemcget`** command options **`-top`** and **`-num_items`** to get the top row index and number of items associated with a table

- Added **`DBHANDLE sort`** command to sort databases

- Added **`DBHANDLE count`** command to return number of records in a database

- Added **`DBHANDLE delete`** command to delete a record in a database

- Added **`dm delete`** command to delete a database

- Added **`DBHANDLE recinfo`** command to return position and unique id of a record

- Added **`%x`** and **`%y`** event substitution specifiers for table events

- Changed **`DBHANDLE set`** command to return index of updated record. ***Incompatible change***

- The `FORMHANDLE getsel` return value sematics have changed for field element types. ***Incompatible change.***

Index

References

---

1 Palm OS Programming for Dummies, Liz O'Hara and John Schettino, IDG Books.

2 Palm Programming, The Developer's Guide, Neil Rhodes and Julie McKeehan, O'Reilly & Associates, Inc.

3 Palm OS Programming Bible, Lonnon R. Foster, IDG Books.

4 Tcl and the Tk Toolkit, John K. Ousterhout, Addison-Wesley.

5 Palm OS Programmers' Companion, Palm Computing.

6 Palm OS SDK Reference, Palm Computing.