# Toucan:

# User Manual

---

A feathered friend for the Palm

---

**Version 1.2.1**

# Table of contents

# Introduction

Toucan (a feathered friend for the Palm) is an cross-platform, Integrated Development Environment (IDE) for creating applications for Personal Digital Assistants (PDA's) running the Palm operating system (Palm OS). The target language is Palm Tcl, a port of Tcl 7.6 with extensions to support applications running in the Palm OS environment. Palm Tcl was developed by Ashok Nadkarni[1]. Toucan is written in Tcl/Tk and can run on both the Gnu/Linux and Microsoft Windows operating systems.

Toucan allows the user to graphically design the layout and properties of Palm resource elements used to create user interfaces and static data. These resources are combined with external data files and the Palm Tcl library by the Pilrc and Par resource generation utilities to form a program that can then be executed on a Palm device or Pose (Palm operating system emulator). A built-in editor is available to allow the user to enter text for data strings and Palm Tcl scripts. When editing Palm Tcl scripts, the editors provide syntax highlighting. The user can also design the launcher icon bitmaps associated with the application. A pop-up window is provided to configure and edit the bitmaps. The user can set up configuration properties for the Palm resource generation utilities. The defined resource elements, icon definitions, and utility configurations can be saved in a Toucan project file (`.tpj`) for later retrieval.

# What's New

Version 1.2 of Toucan contains several enhancements to features found in Version 1.1 and also some new features. Of course, efforts have been made to remove bugs discovered in Version 1.1. The Version History in Appendix A provides a complete list of all changes within Toucan 1.2. Below is a short summary of the enhancements and new features:

- Support for the `STRING` resource is now available. Buttons are now available for creation of arbitrary string resources and those dedicated for use as Palm Tcl scripts.
- A built-in, multiple document interface (MDI) editor is available to support the editing of `STRING` resources. When editing Palm Tcl scripts, the editors provide syntax highlighting. Standard editing functions are available, through both hot keys and an **Edit** menu. Editor characteristics, such as font type, size, and color, can be changed by the user. The editor content can be stored in an external file (mainitaining compatibility with earlier version of Toucan) or embedded within the Toucan project file and the resource script file processed by PilRC. Each editor window has document change detection to prevent loss of unsaved data. The MDI editor is based on the ctext megawidget (version 2.5.1) developed by George Peter Staplin enhanced to provide electric brace, parenthesis, and quotes matching and text change detection.
- The **File** menu has been expanded to with entries for loading and saving documents contained in the MDI editor. When exiting Toucan, the user will be prompted to save editor documents that need to be saved to external files. This is also done when a Palm program resource is generated.
- The **Project** menu (under the Options menu) has been expaned to include an option to automatically load files containing `STRING` resources that are referenced by the Toucan project. When the project file is loaded, the referenced files will be loaded into separate windows of the MDI editor.
- Modal Form elements visualization is better supported. The 'info' icon on a modal Form is displayed on the right-hand side of the titlebar only when a `STRING` resource is selected on the **Help ID** property for the Form. The **Help ID** property selection is located on the Form element properties dialog window.

# System Requirements

Toucan runs on both the Gnu/Linux and Microsoft Windows operating systems. In the Gnu/Linux environment, the Linux kernel should be at least version 2.2 or greater. Toucan was developed and tested on a Slackware Linux 8.0

system, though any comparable Gnu/Linux system should be sufficient. Toucan was tested on Microsoft Windows 98 and 2000 systems and should also run on Windows 95, NT, and XP as well.

# Installation

The Toucan IDE consists of a number of components. Each component provides an essential functionality for Toucan. Some components are integrated into the main Toucan application, while others are launched separately via system executive calls. Figure 1 shows these components and how they interact. Below are instructions for the installation of each component.

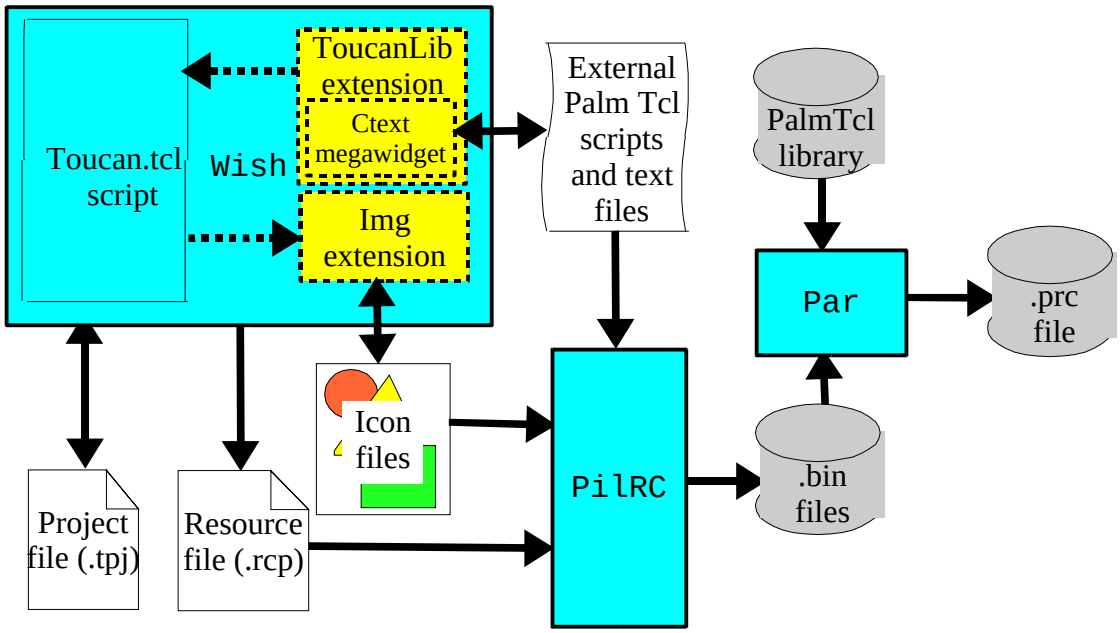


*Figure 1. A block diagram of the development tools and Palm resources used to create a Palm Tcl application (.prc file). Toucan integrates various Tcl extensions (yellow boxes) within the windowing shell (wish) along with the command line utilities PilRC and par (blue boxes) . Toucan creates and manages multiple support files (white boxes) that are used to create the binary files that are combined with the Palm Tcl library to form the Palm Tcl application.*

## *Tcl/Tk*

The Tcl/Tk windowing shell, `wish`, is used to create Toucan's graphical user interface. Toucan requires Tcl/Tk 8.3.4 or later to be installed. If the required version of Tcl/Tk is not installed on the computer, it can downloaded from the ActiveState Tcl website at http://tcl.activestate.com. Pre-compiled binary installations of ActiveTcl are available for both Gnu/Linux and MS Windows. The source code for Tcl/Tk is also available if compiling locally is preferred. Follow the installation instructions that come with the distribution you choose.

## *Img extension*

The Img extension, developed by Jan Nitjmans, is also required by Toucan to support reading and writing BMP format graphic files. If the ActiveTcl distribution mentioned above has been downloaded and installed, then the Img extension installed as well. Source code and MS Windows binaries for Img are available at http://members1.chello.nl/~j.nijtmans/img.html. Follow the installation instructions provided with the Img distribution.

## PilRC and par utilities

The PilRC and par utilities are used by Toucan to create the Palm applications. PilRC (Pilot Resource Compiler) is an application that takes a resource script file and generates one or more binary resource files that are used when developing for the Palm computing platform. PilRC was developed by Wes Cherry and Aaron Ardiri[2]. Toucan is designed to be used with PilRC version 2.8 or greater. Source code and executable binaries for PilRC can be downloaded from http://www.ardiri.com/index.php?redir=palm&cat=pilrc. Follow the installation instructions provided with the PilRC distribution, making sure that the `pilrc` executable binary is placed in a directory that is contained in the operating system's path variable.

The `par` (Palm database archiver) utility creates and manipulates PalmOS database (.pdb) and resource (.prc) files. While par is designed to be a Swiss army knife for managing these files, only one specific functionality of par is used to combine the files generated by PilRC with the Palm Tcl library file to create a Palm application. Par was developed by David Williams. Toucan is designed to be used with par version 00.05.01 or greater. Source code for par can be downloaded from http://www.djw.org/product/palm/par/. Follow the installation instructions provided with the par distribution, making sure that the `par` executable binary is placed in a directory that is contained in the operating system's path variable.

## Palm Tcl library

The Palm Tcl library contains the Palm Tcl interpreter and other supporting functions to allow Palm Tcl scripts to run on the PalmOS. The Palm Tcl library must be combined with the resource files generated by PilRC form a complete Palm application. Toucan supports Palm Tcl version 0.3 or greater. The Palm Tcl package can be downloaded from http://palm-tcl.sourceforge.net/. The directory in which the Palm Tcl library file, `Palm Tcl.prc`, is stored is not critical.

George Peter Staplin's ctext megawidget, version 2.5.1, is the foundation of an embedded programmer's editor with syntax highlighting and line numbering. The editor is used to create `STRING` resources, which can contain either regular text strings or Palm Tcl scripts. The megawidget is incorporated into the ToucanLib library.

## Toucan program

Installation of the Toucan application itself is a two-step operation. First, place the main Toucan program, `toucan.tcl`, into a directory which is contained in the operating system's `path` variable. For example, on Gnu/Linux systems, `/usr/local/bin` could be used. On MS Windows systems, `C:\WINDOWS\COMMANDS` could be used. Any directory is acceptable as long as it is included in the `path` system variable. Alternately, a symbolic link and launcher icon can be placed on the desktop that points to `toucan.tcl`. It is recommended that this method be used to launch the Toucan application on Windows systems. Sample launcher icons for Windows (`toucan.ico`) and GNU/Linux (`toucan_icon.gif`) are provided with the Toucan distribution in the toucanLib's `graphics` directory.

The second step depends upon the operating system that Toucan is running on. For GNU/Linux, use a text editor to open the file `toucan.tcl`. The first line of the file is the windowing shell (`wish`) launch command. By default, this is `#!/usr/local/bin/wish`. The launcher command should be modified to reflect the actual path to the Tcl/Tk windowing shell that Toucan will use. For MS Windows, make sure that the file association exists between the .tcl extension and the execution of the Tcl/Tk windowing shell. For example, the association would cause execution of the program `C:\Tcl\bin\wish83.exe.` If Tcl/Tk was installed from a binary distribution, such as ActiveTcl, this is done automatically.

## ToucanLib

The Toucan library extension (`toucanLib`) contains supporting code and data for the operation of the main Toucan application. Placing these supporting materials into a Tcl extension allows them to be accessed via the Tcl

package command. The Toucan application will find toucanLib if the library is installed in one of the extension storage directories specified by the Tcl `auto_path` variable. The toucanLib extension is installed by copying the toucanLib1.2 directory and its contents to one of the directories specified by the `auto_path` variable. This may require system-level privileges, so consult your system administrator if necessary. Typical values found in `auto_path` are `/usr/local/lib` on Gnu/Linux systems and `C:\Tcl\lib` on MS Windows systems.

# Toucan Philosophy

Toucan is designed to integrate the tasks required to develop PalmOS applications based on Palm Tcl. Palm Tcl cannot form a working application by itself. It relies upon user-defined Palm resources to create the user interface. Some of the resources consist of forms and menus that are used to invoke Palm Tcl scripts. The definition and layout of these resource elements are defined by the programmer in a resource script file (.rcp). Each resource element possesses a unique resource identification number (ID). These resource IDs are used by the Palm Tcl scripts to interact with the resource elements.

Toucan allows the user to graphically design the layout and properties of each resource element. The layout represents the positioning and properties of each resource element, not necessarily the actual appearance of the application as it runs. Most notably, Form objects marked as `UNUSABLE` are still displayed so that their positions are known.

As resource elements are created, they are automatically assigned resource ID numbers by Toucan. These IDs are easily retrieved by the user to place into Palm Tcl scripts. Whenever a resource element is deleted by the use, its assigned ID is returned to the available pool of IDs to be used by Toucan when another element is created.

In addition to the `FORM` and `MENU` resource elements, Toucan supports the creation of the `ICON` , `BITMAP`, and `STRING` resource elements. ICON resources describe bitmaps used by the Palm launcher to run the application. Toucan supports creation of both the larger `ICON` resource bitmap and `SMALLICON` resource bitmap. A bitmap editor is provided by Toucan to allow the user to create these resources. `BITMAP` resouces describe embedded bitmaps used within the Palm application. These graphical elements are displayed in the Palm application using the `FORMBITMAP` Form object. A dialog window is provided to allow the user to select the bitmap files and other are options associated with the BITMAP resources . `STRING` resources describe embedded text strings. `STRING` resources are used to hold Palm Tcl scripts that are evaluated by the Palm Tcl interpreter. Within Toucan, each STRING resource is contained in its own window of the MDI editor.

All of the Palm resource information and settings created within Toucan can be saved to a project file. A Palm application can be created over several development sessions. The content of an entire project be easily moved without concern for misplacing one of the components required to build it.

A screen shot of the main Toucan window is shown in Figure 2. At the top of the window is a menubar. Through the menubar, operational controls and settings of the Toucan IDE can be accessed. The column on the left-hand side of the window contains a set of Palm resource object types. Think of this as a palette from which the user can create the graphical components of the Palm application. The center column in the window contains selectors for resource elements that have been created. Pressing one of these radiobuttons selects and displays the corresponding resource element.The right-hand side of the window is a representation of a Palm PDA screen and control buttons. This facade is used to graphically display the form and menu resource elements that are created for the application. The user may access the resource elements that have been created so they can be edited.

# Toucan Menus

The operational controls and settings of the Toucan IDE are accessed through a menubar at the top of the main window. The menubar has four menus: **File**, **Options**, **Edit**, and **Help**. The operations contained in these menus are described in the following sections.

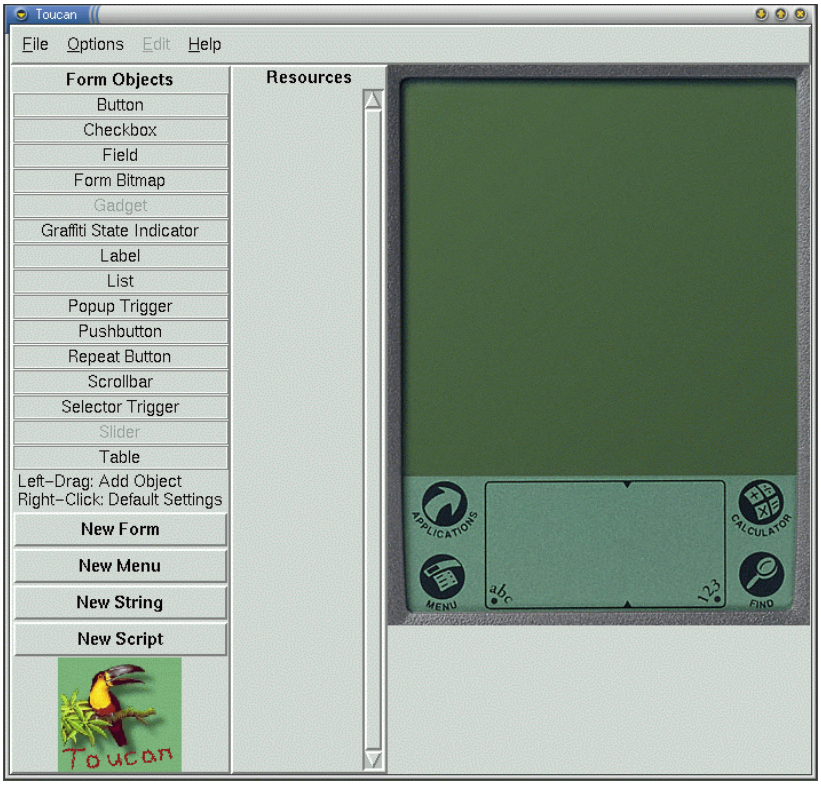


*Figure 2. Toucan main window*

## File menu

The **File** menu provide the user the means to create new Toucan projects as well as save and reload them from files. In addition, the File menu is used to exit the Toucan application. The **File** menu contains the operations **New**, **Load...**, **Save**, **Save As...**, and **Quit**. These operations are described below:

**New** The **New** operation allows the user to create a new Toucan project. Any project that is currently loaded will be deleted. A warning message is displayed to give the user the opportunity to cancel the operation or continue on to create the new project. This warning is shown in Figure 3. If there in no project currently loaded into Toucan, this warning is not displayed.


*Figure 3. Warning message for creating a new project.*

**Load** The **Load** operation is used to load a Toucan project stored in a file into the Toucan IDE. As with the **New** operation, any project that is currently loaded will be deleted. Accordingly, a warning message is displayed to give the user the opportunity to cancel the operation or continue on to load a project. A file selection dialog is displayed to allow the user to select and load a Toucan project file. This dialog window is shown in Figure 4. The user may cancel the operation at this point as well, though the project deleted from the Toucan IDE cannot be restored.

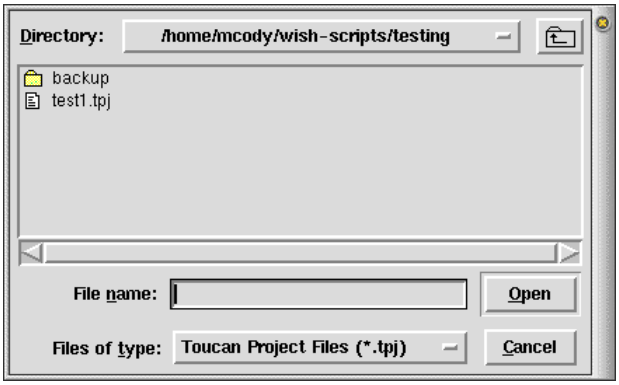


*Figure 4. Project file load dialog.*

**Save** The **Save** operation allows the user to store the Toucan project into a file. If the project is new and has never been saved before, a file selection dialog will be displayed to allow the user to select the directory and name of the file to which the Toucan project will be saved. This dialog window is shown in Figure 5. The user may also cancel the operation at this point. If the project is already assigned to a file, the file selection dialog is skipped.

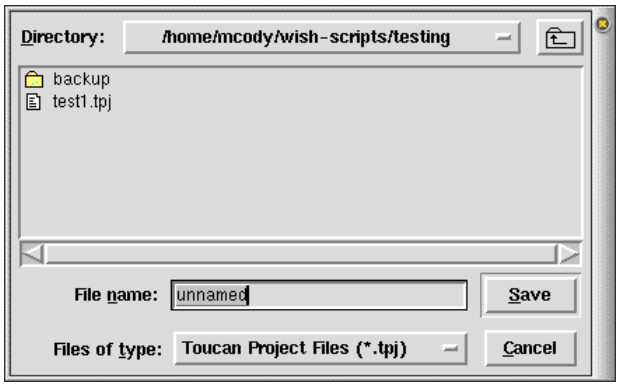


*Figure 5. Project file save dialog.*

**Save As...** The **Save As...** operation allows the user to store the Toucan project into a file other than the file specified when the project was loaded. The same file selection dialog is used as with the **Save** operation.

**Quit** The **Quit** operation is used to exit the Toucan IDE application. When **Quit** is selected a warning is displayed to give the user the opportunity to cancel the operation or continue on the exit the application. This warning is shown in Figure 6.

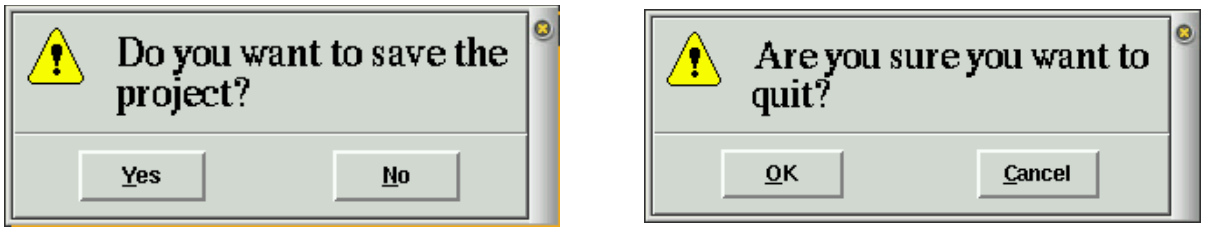


*Figure 6. Toucan program save project and exit warning messages.*

## *Options menu*

In the Options menu, the user can change settings for the generation of Palm applications from within the Toucan IDE. In addition, the user can invoke external routines to create the Palm application. The **Options** menu contains the submenus **Project**, **PilRC**, **Par**, and **Generate**. These submenus are described below:

## Project submenu

The **Project** submenu provides access to general project settings used by Toucan itself. The **Project** submenu contains the operations **Output path...**, **Version...**, **Large icon...**, **Small icon...**, and **Bitmaps...**. The operations available in this submenu are as follows:

**Output path...** Specify the path to the directory which stores the files generated by the project. The value is set through a directory selection dialog window. The directory selection dialog window is shown in Figure 7.

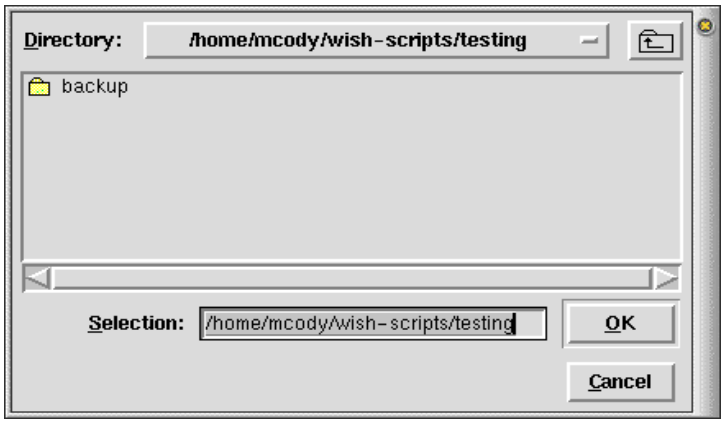


*Figure 7. Output path directory selection dialog.*

**Version...** Specify the value for the Palm `VERSION` resource used in the Palm application generated by the project. The value is set through a dialog window.

**Large icon...** Display the bitmap editor for creating and modifying the large launcher icon. The bitmap editor displays the current appearance of the icon and an fat-bit enlargement (eight-times actual size), which is used to perform the actual editing operations. The user may select the size of the larger icon as 32x32, 32x22, or 22x22 pixels. The bitmap editors dialogs showing 32x32 and 22x22 pixel icons are shown in Figure 8. The user may also select either black or white for coloring the bitmap pixels. Pressing the Color button presents a drop-down menu through which the user can select the desired color. The user colors pixels by placing the mouse cursor on the fat-bit image and pressing the left mouse button. Multiple pixels can be colored by dragging the mouse cursor over the fat-bit image while holding the left mouse button down.

**Small icon...** Display the bitmap editor for creating and modifying the small launcher icon. The bitmap editor is similar to that used for editing large icons. It is shown in Figure 9.

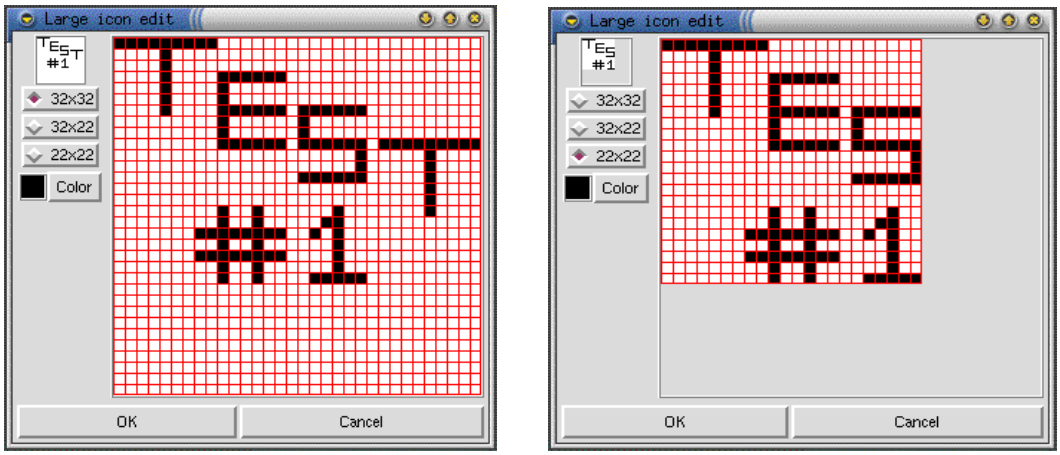


*Figure 8. Bitmap editor dialogs for 32x32 (left) and 22x22 (right) pixel icons.*

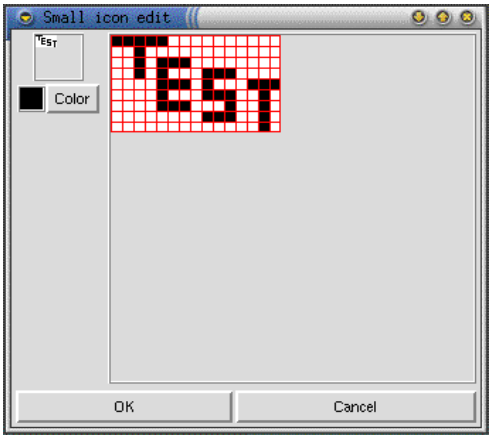


*Figure 9. Bitmap editor dialogs for small (15x9 pixel) icons.*

**Bitmaps...** Allow the create Bitmap elements and specify their attributes. The bitmap properties dialog widow, shown in Figure 10, lists the currently existing Bitmap element IDs and the bitmap files (BMP format) associated with them. The operation of the **View**, **File**, **Type**, and **Compression** controls requires that one on the existing Bitmap elements be selected via a mouse click. The **View** button allows the user to display the image contained in the currently selected bitmap file. The **File** button allows the user to change which bitmap file is associated with the Bitmap ID of the currently selected Bitmap element. A file selection dialog similar to that shown in Figure 4 is displayed to allow the user to select the desired bitmap file. The **Type** menu is used to choose the specific type of Bitmap for the selected Bitmap element (`BITMAP`, `BITMAPGREY`, `BITMAPGREY16`, `BITMAPCOLOR16`, `BITMAPCOLOR`, or `BITMAPCOLOR16K`). The **Compression** menu is used to choose the compression type used for the selected Bitmap element (`NOCOMPRESS`, `COMPRESS`, or `FORCECOMPRESS`). The **Add** button creates a new Bitmap element the file selection dialog described above is used to pick the desired Bitmap file. The **Delete** button deletes the selected Bitmap element. All additions, changes, and deletions do not become permanent until the **OK** button is pressed to dismiss the dialog. Note that when a Bitmap element is deleted, all existing references to it by Form element properties are automatically changed to 'None'. The **Cancel** button abandons all changes that have been made.

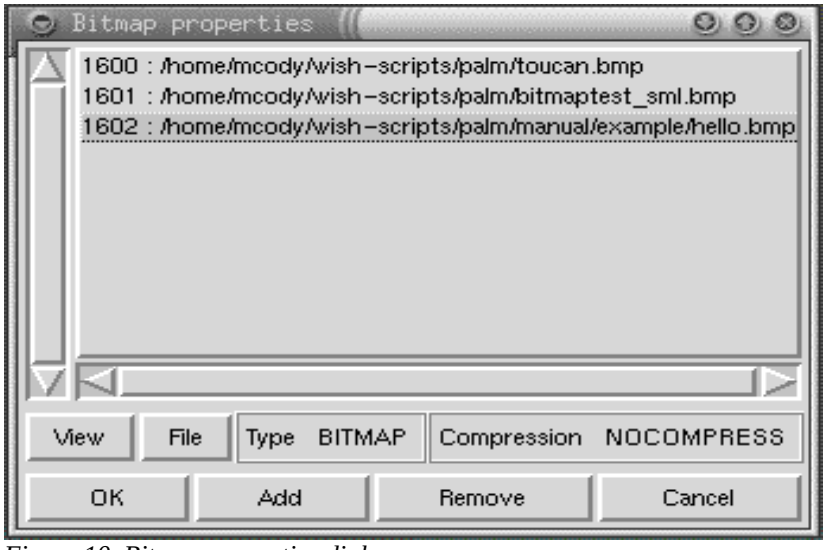


*Figure 10. Bitmap properties dialog.*

**Autoload STRING files** When this option is selected, all files referenced by STRING resource editors will be loaded automatically when the project file is loaded.

## PilRC submenu

The **PilRC** submenu provides access to settings used by the `pilrc` utility. The **PilRC** submenu contains the operations **Language**, **Auto font width**, **Edit menu IDs**, **Little-endian resources**, **PalmRez**, **Quiet output**, **Right-to-left**, **MS-style output**, **Include path(s)...**, **Macro symbol(s)...**, **Resource file output...**, and **Include file output...**. The operations available in this submenu are as follows:

**Language** Select the target language for resource file generation. The available selections in this submenu are:

- **English** The default language, no option inserted.
- **French** Insert the command line option `-L FRENCH`.
- **German** Insert the command line option `-L GERMAN`.
- **Italian** Insert the command line option `-L ITALIAN`.
- **Spanish** Insert the command line option `-L SPANISH`.

**Auto font width** Select the font width style to be used for `AUTO` width calculations. The available selections in this submenu are:

- **Default** Standard font widths are used.
- **Chinese (Big5)** Insert the command line option `-F5`.
- **Chinese (GB)** Insert the command line option `-Fg`.
- **Hebrew** Insert the command line option `-Fh`.
- **Japanese** Insert the command line option `-Fj`.
- **Korean (Hanme)** Insert the command line option `-Fkm`.
- **Korean (Hantip)** Insert the command line option `-Fkt`.

**Edit menu IDs** Enable or disable the command line option `-allowEditID` to allow the use of 'edit menu' ID values (10000-10007 inclusive).

**Little-endian resources** Enable or disable the command line option `-LE32` to generate Little Endian 32 bits compatible resources.

**PalmRez** Enable or disable the command line option `-PalmRez` to generate resources with the PalmRez option.

**Quiet output** Enable or disable the command line option `-q` to produce less noisy output.

**Right-to-left** Enable or disable the command line option `-rtl` to allow support of right to left text.

**MS-style output** Enable or disable the `-V` command line option to generate M$ (VS-style) warning/error output (default is GNU-style).

**Include path(s)...** Insert one or more instances of `-I` *`INCLUDE PATH`* into the command line. Search each *`INCLUDE PATH`* when looking for include or bitmap files. The include path dialog window is similar to the directory selection dialog shown in Figure 11. Multiple paths can be added through a directory selection dialog similar to that shown in Figure 7.

**Macro symbol(s)...** Insert one or more instances of `-D` *`MACRO[=VAL]`* into the command line. Define a pre-processor symbol with an optional value. The macros defined with the `-D` option can be referenced in `#ifdef` statements in the resource file for conditional compilation. The macro definition dialog window is similar to that shown in Figure 11. Multiple macros can be added through a macro/value pair entry dialog window. NOTE: if no value is specified, the symbol will given the value of 1.

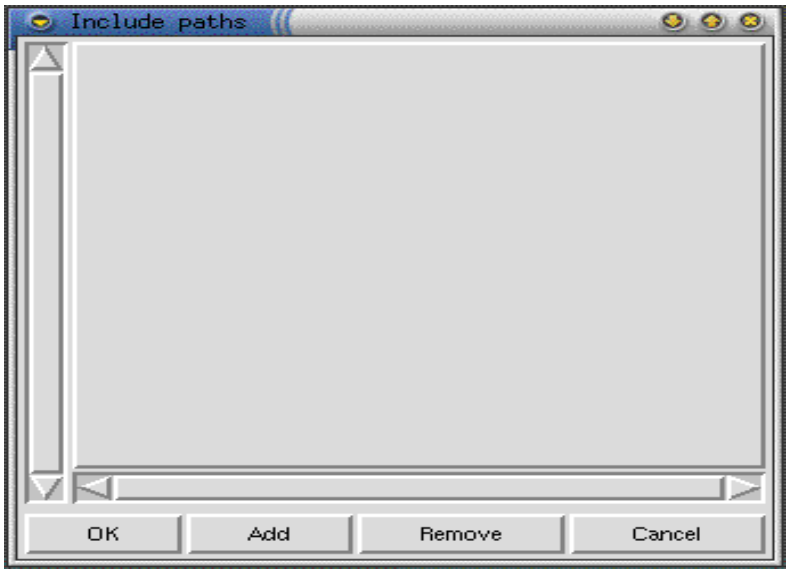


*Figure 11. Include path dialog window.*

**Resource file output...** Output a `.res` file specifying all the resources emitted by PilRC by using the `-R` *`RESFILE`* command line option. The value of *`RESFILE`* is set via an entry dialog window.

**Include file output...** Output a header (`.h`) file containing auto-generated resource item IDs for resource items that were defined without an ID previously. The header file is specified by inclusion of `-H` *`INCFILE`* on the command line. NOTE: If `-H` is not specified, then undefined IDs are considered errors.

## Par submenu

The **Par** submenu provides access to settings used by the `par` utility. The **Par** submenu, shown in Illustration 8a, contains the operations **Application name...**, **Creator ID...**, and **Palm Tcl library path...**. The operations available in this submenu are as follows:

**Application name...** The name of the Palm Tcl application which is displayed underneath or beside the launcher icon on the Palm applications display. The value is set via an entry dialog window.

**Creator ID...** The unique, four-character creator identifier used by all Palm applications and their associated databases. The value is set via an entry dialog window.

**Palm Tcl library path...** Specify the path to the directory which stores the Palm Tcl library file. The value is set through a directory selection dialog window. The directory selection dialog window is similar to the one shown in Figure 7.

## Generate submenu

The **Generate** submenu is used to create Palm Tcl applications and their components. This submenu, contains the operations **Target Palm resource file**, **Resource script file only**, **Large Icon file only**, and **Small icon file only**. The operations available in this submenu are as follows:

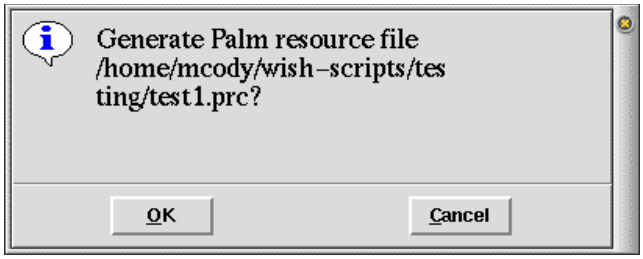


*Figure 13. Alert message for Palm application generation.*

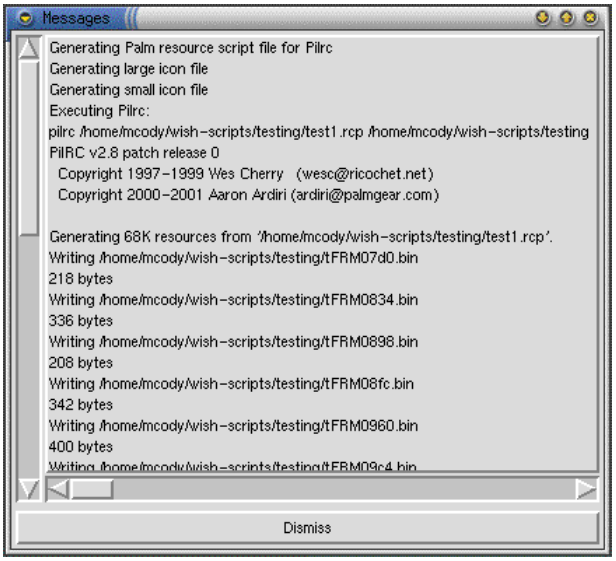


*Figure 12. Messages dialog showing generation of Palm resource file.*

**Target Palm resource file** — Generate the Palm Tcl application (`.prc`) file that will run on the Palm. As part of this operation, the resource script and icon bitmap files will be generated. A alert message, as shown in Figure 13, will be presented to allow the user to either continue or abort the operation. If allowed to continue, Toucan will display a message dialog window listing the messages generated by the `pilrc` and `par` programs. An example is shown in Figure 12.

**Resource script file only** — Generate the resource script (`.rcp`) file that will be processed by `pilrc`. A warning message, similar to that in Figure 13, will be presented to allow the user to either continue or abort the operation.

**Large icon file only** — Generate the large icon bitmap (`.bmp`) file that will be displayed by the Palm launcher. A warning message, similar to that in Figure 13, will be presented to allow the user to either continue or abort the operation.

**Small icon file only** — Generate the small icon bitmap (`.bmp`) file that will be displayed by the Palm launcher. A warning message, similar to that in Figure 13, will be presented to allow the user to either continue or abort the operation.

## *Edit menu*

This **Edit** menu is enabled whenever a STRING resource editor is displayed. The operations affect the currently selected editor. Text placed into the cut buffer while one editor is selected can be pasted into any other editor inside Toucan. The text can also be placed into an external application if it uses the cut buffer for editing operations. The menu provides sever al common text editing functions. The **Edit** menu contains the operations **Cut**, **Copy**, **Paste**, **Delete.**, **Select all**, **Search/Replace...**, and **Preferences..**. These operations are described below:

**Cut** — Remove the text selected in the displayed edtor and put it into the cut buffer.

**Copy** — Copy the selected text in the displayed editor and put it into the cut buffer.

**Paste** — Paste the text in the cut buffer into the displayed editor at the current postion of the text cursor.

**Delete** — Remove the text selected in the displayed editor.

**Select all** — Select all text in the displayed editor.

**Search/Replace...** — Display the text search and replace dialog window. Options are available to select case sensitivity, regular expression versus exact text string matching, and the direction of the search through the text body. Text replacement for the next search or for all matches can be performed as well. This dialog is shown in Figure 14.

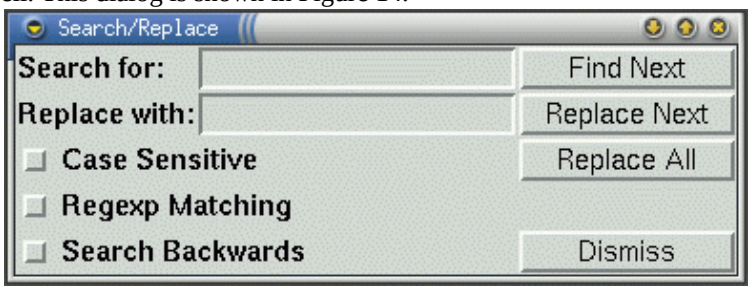


Figure 14. Editor search and replace dialog window.

**Preferences...** — Display the text editor preferences dialog window. The font used for all editors can be selected as well as the colors applied for the syntax highlighting. The color used and blink rate for electric brace matching can also be set. An example text window is provided on the right hand side of the dialog to allow the user to experiment with the settings. This dialog is shown in Figure 15.

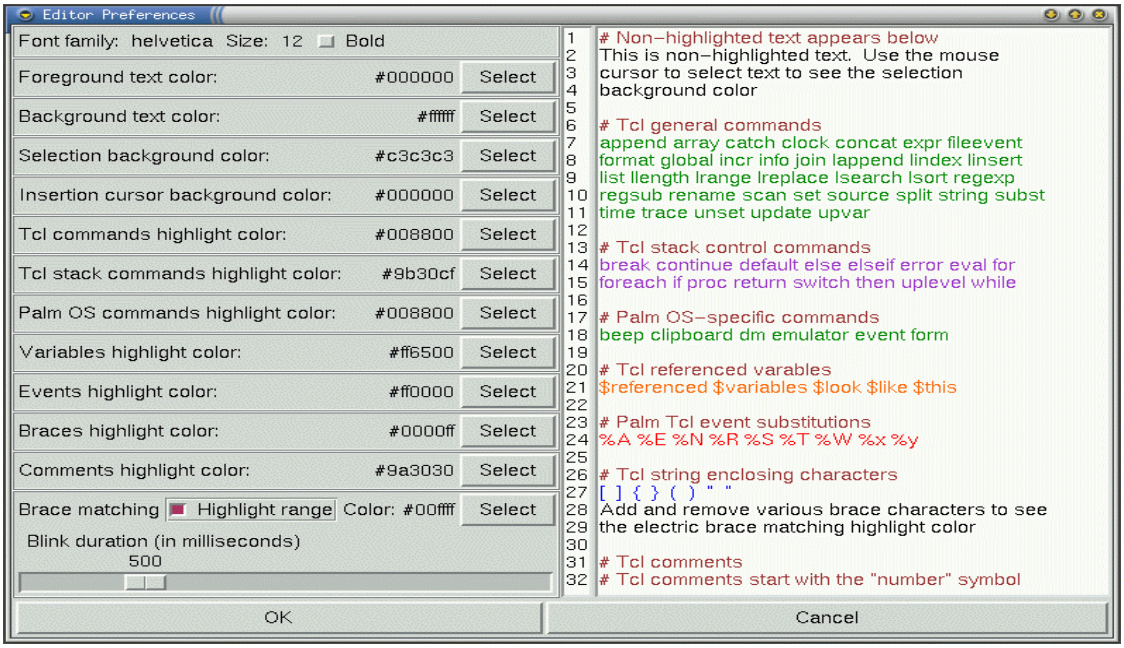

Figure 15. String resource editor preferences dialog window.

### *Help menu*

Currently, the **Help** menu only contains one entry; **About...**, which lists the application name, its author, and a copyright notice. In a future version of Toucan, an online help system will be added.

## Creating Palm Forms Graphically

Toucan provides the user the ability to create Palm Tcl applications graphically. That is, the user can see what the user interface will look like (approximately) before the application is generated and run on a Palm device or Pose. Doing this enables a speed-up of the development process. Still, the user needs to gain an understanding of Palm resources and how they are used to create Palm applications.

### *Palm resources*

Palm resources are static entities of different types that are contained in a running Palm application. The information held in the resources can specify user interface components, support data used by the application, or support data used by the Palm OS to gain knowledge about the application. Each resource is known by the application and the Palm OS by a unique resource identification number (ID) assigned to it during the design of the application. When a programmer designs a Palm application, he must be aware of which resource types are assigned which IDs. If a program attempts to use an ID to access a resource which doesn't exist or is of the wrong type, then unexpected results could occur.

Toucan helps to solve this problem by automatically assigning IDs to resources as they are created. Each ID is guaranteed to be unique due to the approach Toucan uses to assign the IDs. This approach is to allocate ranges of numbers for use by different types of resource elements. As each new resource is created, the ID it is assigned is taken from the range allocated for that resource type. If a resource is deleted, that ID becomes available for reuse when another resource of that type is created. Table 1 shows the ranges to which resources are assigned.

| Resource | Assigned range | Increment between instances |
|---|---|---|
| BITMAPS (all types) | 1600 - 1699 | 1 |
| FORM | 2000 - 5900 | 100 (maximum of 40 FORMs |
| FORM objects | 2x01 - 5x99 | 1 (maximum of 99 per FORM) |
| GROUPS | 1500 - 1599 | 1 |
| ICON | 1000 | Only one instance per application |
| MENU | 6000 - 9900 | 100 (maximum of 40 MENUs) |
| MENUITEM | 6x01 - 9x89 | 1 (at most 6 PULLDOWNs, 14 MENUITEMs in each) |
| SMALLICON | 1001 | Only one instance per application |
| STRING (arbitrary) | 1700 - 1799 | 1 |
| STRING (scripts) | 9999 - 9900 | -1 |
| VERSION | 1 | Only one instance per application |

*Table 1. Resource ID assignments*

## *Resources supported by Toucan*

Toucan currently supports the following Palm resources: BITMAP, BITMAPCOLOR, BITMAPCOLOR16, BITMAPCOLOR16K, BITMAPGREY, BITMAPGREY16, FONT, FORM, ICON, MENU, SMALLICON, STRING, and VERSION. All properties of the FORM and FORM resource objects can be adjusted. The FORM resource objects currently supported by Toucan are: BUTTON, CHECKBOX, FIELD, FORMBITMAP, GRAFFITISTATEINDICATOR, LABEL, LIST, POPUPTRIGGER, PUSHBUTTON, REPEATBUTTON, SCROLLBAR, SELECTORTRIGGER, and TABLE. All properties of these FORM resource objects can be adjusted. The APPLICATIONICONNAME resource is indirectly supported in that the application name string is supplied to the par utility when the Palm Tcl application is created. The POPUPLIST resource is indirectly supported in that a **List ID** property is included with the POPUPTRIGGER resource configuration dialog to indicates that the POPUPTRIGGER will be used to display a LIST resource via a POPUPLIST.

## *Forms and Form objects*

Forms and Form objects are the primary means by which the user interacts with Palm applications. Think of them in the same way as with dialog windows in graphical user interfaces of applications running on X Windows under GNU/Linux or on Microsoft Windows. The placement and content of Form and Form object determine the essential character of a Palm application.

Toucan provides the means to quickly create and manage Forms and Form objects. The general characteristics can be set up through defaults. The properties and positioning of specific resource elements are also handled.

## Setting defaults

When a Form or Form object is created, it is assigned default values for its properties. The user can change the default values that are applied by accessing the default property dialogs for the Form or Form objects. The default property dialogs are accessed by positioning the mouse cursor over the Form button or appropriate Form object label and clicking with the right mouse button. The dialog window for the Form or Form object properties will be

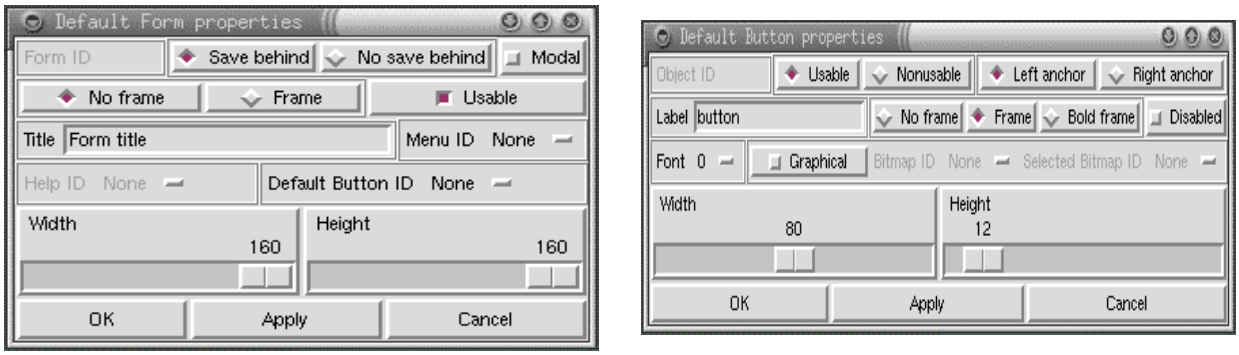


*Figure 16. Default properties dialogs for the Form element (left) and the Button object (right).*

displayed. The Form and Button default property dialogs are shown in Figure 16. Note that the Graffiti State Indicator does not have a default properties dialog since it has no properties to change.

Some properties in the Form and Form object dialogs relate to resource IDs. For example, the Menu ID property in the Form dialog allow the specification of a Menu resource in the project that is tied to the Form resource. All resource IDs are selected via pick lists. As resources of various types are created, they are made available automatically in pick lists for those resource types within the appropriate dialog windows. In addition, the absence of a selected resource ID is represented by the selection 'None'. Automatic width and height specifiers are available on most Form elements. Whenever the width or height is set to 0, the `AUTO` keyword is inserted into the appropriate location in the resource script (.rcp) file when it is generated.

Once the properties are set to the desired values, pressing the **Apply** button will commit the new default property values. Pressing the **OK** button will commit the new default property values and close the dialog window. Pressing the **Cancel** button will close the dialog window without changing the default property values.

## Using Palm fonts

There are seven standard font types supported by the PalmOS. Some fonts are used to display alphanumeric information; such as memos, appointments, and address book entries. Other fonts are used to display symbolic information; like the graffiti state indicator or the note and alarm indicators in the date book application. These are enumerated in Table 2 below:

| Font ID | Font name | Symbol range | Description |
|---|---|---|---|
| 0 | stdFont | n/a | Standard small font used for writing |
| 1 | boldFont | n/a | Small, bold font that is easier to read |
| 2 | largeFont | n/a | Larger font used for easier reading |
| 3 | symbolFont | 3 through 26 | Various user interface images like check boxes, arrows, and the graffiti state indicator icons |
| 4 | symbol11Font | 0 through 5 | Larger user interface images for check boxes and arrows |
| 5 | symbol7Font | 1 through 4 | User interface images for vertical scrolling |
| 6 | ledFont | 44 through 57 | Calculator-specific font |
| 7 | largeBoldFont | n/a | A thicker, more readable version of the large font |

*Table 2. Palm OS standard fonts*

The character fonts (0, 1, 2, and 7) are normally used for data entry and display for the user. When any of these fonts are selected through one of the Form object properties dialogs, it is presumed that multiple characters will be

used in strings. These character strings will be placed in either a Label or List entry displayed in the properties dialog or entered interactively through a `FIELD` Form object. The literal character string that are entered will appear as they would within the corresponding Form object.

Under most programming situations, the symbol fonts (3, 4, 5, and 6) are used as individual characters within one of the button or label Form objects (though they can be used in any Form object that has a font property). The valid range of character code values in the symbol fonts necessitates specifying individual symbols by their character codes rather than explicitly. When any of the symbol fonts are selected through one of the Form object properties dialogs, it is presumed that individual characters will be used. If the character string entered in an Label entry represents the character code for a valid symbol in the selected font, then the corresponding symbol image will be displayed in the corresponding Form object. If it does not, then the literal character string will be displayed within the corresponding Form object. Note that `FIELD` and `LIST` Form objects will always display literal character strings even when symbol fonts are selected. This is a limitation of Tk widgets rather than the actual performance of the PalmOS Form objects.

## Creating Forms

Form resources are created by pressing the **New Form** button on the lower left-hand side of the Toucan main window. The `FORM` appears on the simulated Palm display on the right-hand side of the Toucan main window. The characteristics of the Form are set according to the current default values for the Form resource properties. At the same time, a new radiobutton appears under the **Resources** column on the center of the Toucan main window. The radiobutton is labeled **Form *XX00***, where ***`XX00`*** is the resource ID assigned to the Form resource. This button can be pressed to select the resource for display. Any other Form or Menu resource shown on the Palm display is hidden. Only one Form or Menu resource is shown at any time.

## Creating Form objects

Once one or more Form resources are created, Form objects can be inserted into the Forms. This is performed through a drag and drop operation. The mouse cursor is placed over one of the Form object labels listed under the **Form objects** heading on the left-hand side of the Toucan main window. The left mouse button is depressed and a small window will appear, containing a representation of the Form object. Holding the left button down and dragging the mouse cursor causes the Form object to follow the mouse cursor. If the left mouse button is released while the mouse cursor is not directly on a Form resource, a warning beep is issued and the Form object and the small window that holds it are withdrawn. This also occurs if the mouse cursor is on top of a Form object already placed on the Form. Releasing the left mouse button while the mouse cursor is on a Form resource will cause the Form object to be deposited onto the Form resource with its upper left-hand corner positioned at the location of the mouse cursor. The characteristics of the Form object are set according to the current default values for the Form object's properties.

## Repositioning

Once a Form or Form object is placed onto the Palm display, it can be repositioned the Palm display. The positioning of Forms are restricted by the size of the Form resource. A Form that is 160 pixels high by 160 pixels wide cannot be repositioned because it fills the entire Palm display region. Form objects within a Form resource can only be repositioned within the current bounds for the Form resource.

Repositioning is accomplished by placing the mouse cursor on the Form or Form object that is to be repositioned. Depressing and holding down the right mouse button activates the repositioning mode. In the upper left-hand corner of the Palm display, a indicator will appear showing the current coordinates of the upper left-hand corner of the Form or Form object being repositioned. An example of this is shown in Figure 17. Moving the mouse cursor while holding the right mouse button down will drag the Form or Form object to a new position. Releasing the right mouse button completes the repositioning operation and dismisses the coordinates indicator. Note that the

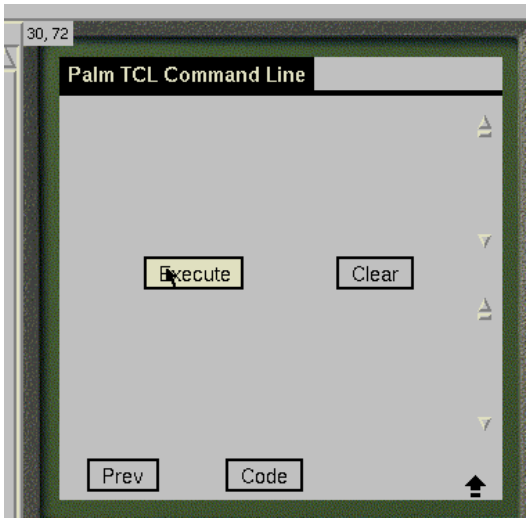


*Figure 17. Example of repositioning a Button object on a Form element.*

coordinates for Form repositioning is relative to the Palm display, while the coordinates for Form object repositioning is relative to the Form that contains the Form object.

## *The resource control menu*

Each resource element that is created contains information that need to be managed. Accessing this information also aids the development of Palm Tcl scripts that will control the resource elements. This information is accessed through a resource control menu that can be displayed by the user. The resource control menu is displayed by placing the mouse cursor on the resource element and then performing a double-click with the right mouse button. A successful double-click will display the resource control menu on top of the resource element with the menu's upper left-hand corner positioned on the mouse cursor.

An example of a displayed resource control menu is shown in Figure 18. The resource control menu has four entries: **Dismiss**, **ID = *XXXX*** (where ***XXXX*** is a resource ID number), **Properties**, and **Delete**. Note that `GRAFFITISTATEINDICATOR`s do not possess resource IDs or have properties to modify. When the resource

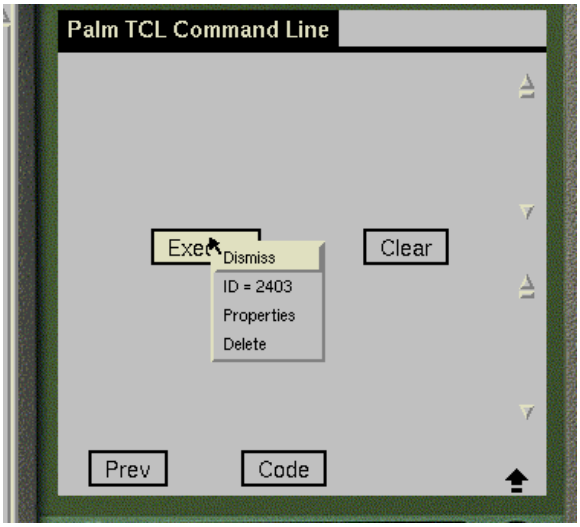


*Figure 18. A resource control menu posted for the underlying Button element.*

control menu is displayed, the **ID = *XXXX*** menu entry is replaced by **No ID**. That menu entry and the **Properties** menu entry are disabled.

The **Dismiss** entry causes the resource control menu to dismiss with no additional actions taken. It is provided simply as a means of dismissing the resource control menu without having any actions taken.

## Obtaining resource IDs

The **ID = *XXXX*** entry shows the assigned ID number for the selected resource element. If this menu entry is selected, the ID number is placed into the text cut buffer and the resource control menu is dismissed. This allows for quick, accurate copying of resource IDs so they can be inserted into an editor containing a Palm Tcl script.

## Modifying properties

The **Properties** menu entry is used to display a properties dialog window for the selected resource element. The properties dialogs are similar to the default properties dialogs mentioned earlier. The differences are that ID of the resource element is displayed in the dialog and the property values are specific to the selected resource element, not a global default. Applying any changes only affect the selected resource element and no others

## Deleting resource elements

The **Delete** menu entry is used to allow removal of a resource element. If this menu entry is selected a warning message is displayed to allow the user to accept or cancel the operation. Once the resource element is deleted, it cannot be recovered. This fact should be carefully considered when deleting Form or Menu resources. The resource elements that reside within these resources will be lost as well.

### *Menu elements*

Menus are the means by which the user accesses transient controls and operations on the Palm application. They work in the same way as menus in graphical user interfaces of applications running on X Windows under GNU/Linux or Microsoft Windows.

## Creating menus

Menu resources are created by pressing the **New Menu** button on the lower left-hand side of the Toucan main window. The `MENU` appears at the top of the simulated Palm display on the right-hand side of the Toucan main window. The characteristics of the Menu are set up for a minimal configuration: A menubar with a single pulldown, which holds a single menuitem. At the same time, a new radiobutton will appear under the **Resources** column on the center of the Toucan main window. The radiobutton is labeled **Menu *XX00***, where ***XX00*** is the resource ID assigned to the Menu resource. This button can be pressed to select the resource for display. Any other Form or Menu resource shown on the Palm display is hidden. Only one Form or Menu resource is shown at any time.

Unlike Form resources, Menus cannot be repositioned. They always reside at the top of the Palm display. Therefore, the click and drag functionality associated with Form and Form objects is not available with Menu resources. Menus do possess IDs, properties, and can be deleted. Therefore, a double-click with the right mouse button will display a resource control menu on Menu resources as it does with Forms and Form objects.

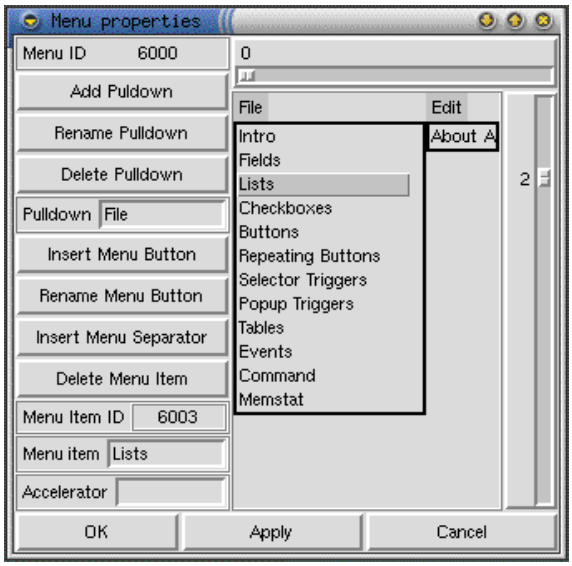


*Figure 19. Menu properties dialog.*

## Modifying menus

When the **Properties** menu entry on the resource control menu is pressed, the Menu resource properties dialog window is displayed. This is shown in Figure 19. With the Menu resource properties dialog window, the user can add, delete, modify, and reposition Pulldowns and individual Menuitems in the selected menu. At any time, the user may press the **OK** button to comit the modifications to the menu and dismiss the dialog. Pressing the **Apply** button comits the modifications but does not dismiss the dialog. Pressing the **Cancel** button discards all modification made to the Menu that have been performed after the last comit.

## Adding pulldowns

In the **Pulldown** entry field, replace the current name string with the name of the new Pulldown you wish to add. Now press the **Add Pulldown** button to add the new Pulldown. It will added to the right side of the other Pulldowns currently displayed. If the **Add Pulldown** button is pressed when the entry field is blank, an error message is displayed alerting the user to this condition. At most, six `PULLDOWN` Menu elements occur on a Menu. Attempting to add more results in an error message notifying the user that the maximum has been reached.

## Deleting pulldowns

Highlight any one of the Menuitems on the Pulldown you want to delete. Press the **Delete Pulldown** button. If there is only one Pulldown on the Menu, an error message is displayed to tell the user that at least one Pulldown must be on the Menu. If there are more than one Pulldown on the Menu, a warning message is displayed to allow the user to accept or cancel the Pulldown deletion. Once the Pulldown is deleted, it cannot be recovered in the Menu Properties dialog. This fact should be carefully considered when deleting Pulldowns. The Menuitems that reside within Pulldown will be lost as well.

## Modifying pulldown names

Highlight any one of the Menuitems on the Pulldown you want to rename. Enter the new name of the Pulldown in the **Pulldown** entry field. Press the **Rename Pulldown** button to complete the task. If the **Rename Pulldown** button is pressed when the entry field is blank, an error message is displayed alerting the user to this condition.

## Repositioning pulldowns

Highlight any one of the Menuitems on the Pulldown you want to reposition. Place the mouse cursor on the slider above the Menu display area. Press and hold down the left mouse button and move the mouse to shift the slider handle and the relative position of the Pulldown with respect to the other Pulldowns. Release the mouse button to complete the task.

## Adding menuitems

Highlight the Menuitem on the Pulldown where you want to insert a new Menuitem. In the **Menu item** entry field, replace the current name string with the name of the new Menuitem you wish to add. Optionally you can place a value into the **Accelerator** entry field (usually a single character). Make sure that the highlighted Menuitem has remained that way or the first step will have to be repeated. Now press the **Insert Menu Button** button to insert the new Menuitem. It will take the place of the highlighted Menuitem, shifting it below the new one. If the **Insert Menu Button** button is pressed when the entry field is blank, an error message is displayed alerting the user to this condition. At most, fourteen `MENUITEMS` can occur on a Pulldown. Attempting to add more results in an error message notifying the user that the maximum has been reached Menuitem `SEPARATOR`s can also be inserted into Pulldowns by pressing the **Insert Menu Separator** button.

## Deleting menuitems

Highlight the Menuitem you want to delete. Press the **Delete Menu Item** button. If the Menuitem is the only one on the Pulldown, an error message is displayed to tell the user that at least one Menuitem must be on the Pulldown. Once the Menuitem is deleted, it cannot be recovered in the Menu properties dialog.

## Modifying menuitem names

Highlight the Menuitem to be renamed. In the **Menu item** entry field, replace the current name string with the name of the new Menuitem you wish to add. Optionally, the value in the **Accelerator** entry field can be changed (usually a single character). Make sure that the highlighted Menuitem has remained that way or the first step must be repeated. Now press the **Rename Menu Button** button to complete the task. If the **Rename Menu Button** button is pressed when the entry field is blank, an error message is displayed alerting the user to this condition.

## Repositioning menuitems

Highlight the Menuitem to be repositioned. Place the mouse cursor on the slider to the right of the Menu display area. Press and hold down the left mouse button and move the mouse to shift the relative position of the Menuitem with respect to the other Menuitems on the Pulldown. Release the mouse button to complete the task.

## *String resources*

The `STRING` resource contains an arbitrary text string that can be accessed by a Palm application. The text string assigned to a `STRING` resource can either be a sequence of characters present in the resource declaration or a reference to a file containing the text. Palm Tcl uses `STRING` resources to hold Tcl scripts that are executed by the interpreter. Palm Tcl requires that a `STRING` resource with the ID of 9999 exist in the application. The script contained in this resource is executed first by the Palm Tcl interpreter. Additional scripts can be placed in other `STRING` resources, which can be read by the Palm Tcl interpreter using the Tcl *source* command. Note that any string resource can be sourced into the Palm Tcl interpreter. Script strings are just dedicated for this purpose.

Toucan can create `STRING` resources for use as arbitrary strings or as Palm Tcl scripts. The selected use determines the resource ID assigned to the resource. As shown prevously in Table 1, arbitrary strings are given

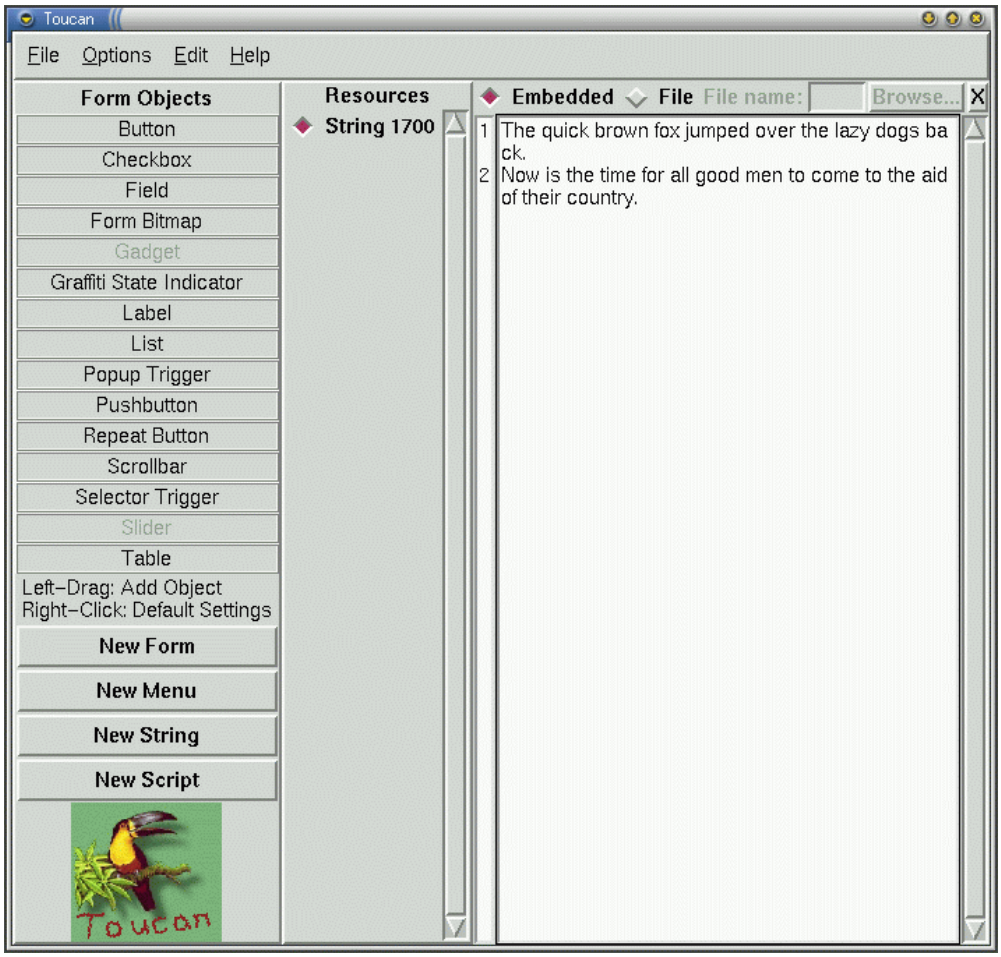

Figure 20. Embedded text editor for an arbitrary STRING resource.

resource IDs starting with 1700 and ending with 1799. Palm Tcl scripts are given resource IDs starting with 9999 and ending with 9900. This reverse progression is done so that the first Palm Tcl script created is always the one first executed by the Palm Tcl interpreter. This `STRING` resource should always be created when a Palm Tcl application is developed.

Either type of `STRING` resource is created by pressing the appropriate button on the Toucan main window. The button, **New String**, creates a STRING resource for an arbitrary string. The button, **New Script**, creates a `STRING` resource that will be a Palm Tcl script. An editor window is created for each resource created so that the user can create and edit the text string or specify the file that contains the text. Figure 20 shows an editor for `STRING` resource 1700. The editors for script strings have the extra enhancement of sytax highlighting. This assists the user in developing Tcl scripts. The string resource editor preferences dialog window in Figure 15 shows the nature of the syntax highlighting for the different Palm Tcl commands.

The options at the top of the editor allow the user to select how the text in the `STRING` resource is stored. If the **Embedded** option is selected, the text is stored within the Toucan project file and within the Palm resource script file when the Palm Tcl application is created. When the **File** option is selected, the text is stored in an external file. The name of the external file can be entered into the **File name** entry field. A file browser dialog is opened when the **Browse** button is pressed. The selected file's path and name are placed into the **File name** entry field. The file contents is loaded and saved using the **Load File** and **Save File** operations, respectively, in the **File** menu. Note that a separate editor can be used to create and edit the string and script files, if desired. All `STRING` resources may be

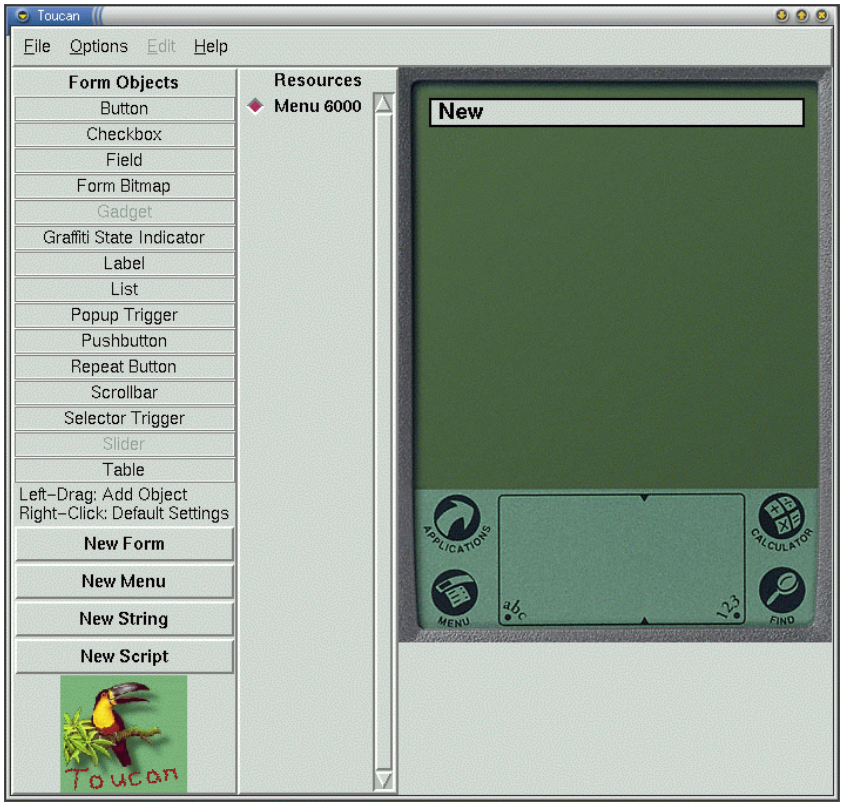


*Figure 21. Initial Menu element created via the 'New Menu' button.*

switched between the embedded and external file options at any time. The **X** button at the far right allows the user to delete the resource. The user is prompted to verify the deletion of the resource .

Whenever the text in an editor is changed, an internal flag is set to indicated that saving is needed for that editor. This only applies to text stored in external files. The flag determines the enabled state of the **Load File** and **Save File** entries in the **File** menu. An active flag also forces a prompt to save the text to a file when exiting Toucan.

# A Simple Sample Example

The following sections describe the creation of a simple Palm Tcl application using the Toucan IDE. This is a three-step process. The first step involves the creation of the user interface that is displayed by the Palm device. This is accomplished graphically through the Toucan IDE. The second step is setting up the application's characteristics used by the PalmOS. The third step is the creation of the Palm Tcl script which drives the user interface.

## *Creating the Palm Tcl user interface*

After starting Toucan, press the button **New Menu** to create a new Menu element, as shown in Figure 21. Menu elements cannot be repositioned, as can Form elements, and are always at the top of the Palm display. Use the mouse to display the resource control menu, and select the **Properties** menu button to display the Menu element properties dialog. The initial contents of the dialog is shown in the left side of Figure 22.

The first change to the Menu element dialog is renaming the Pulldown object labeled New. This is accomplished by first placing the mouse cursor over the Menuitem object labeled '<new>' and pressing the left mouse button. Doing this causes the menu item to be selected. The values of the menu Pulldown title and Menuitem label will be

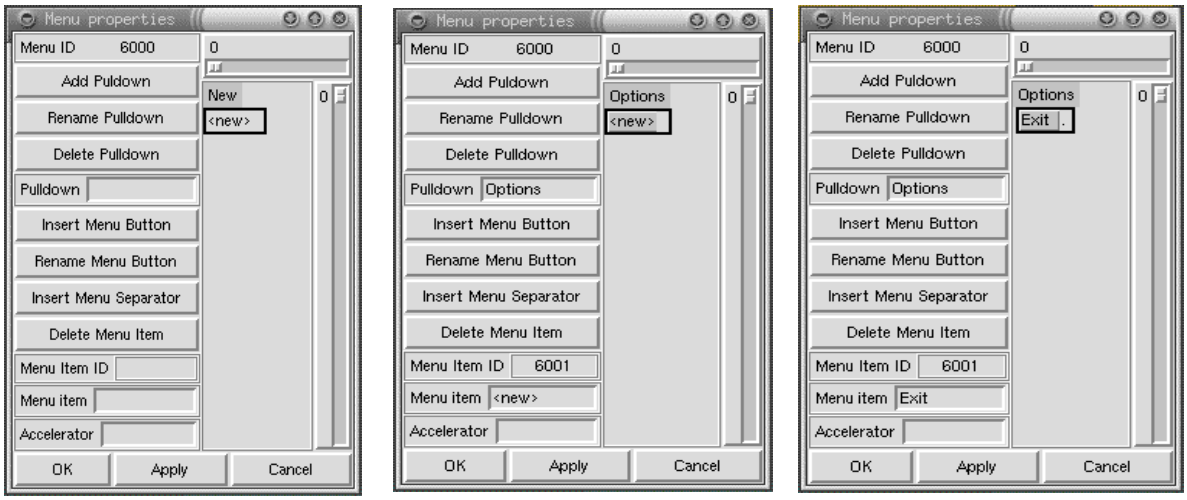


*Figure 22. Initial Menu element dialog (left), after renaming a Pulldown entry from 'New' to 'Options' (center) and renaming a Menuitem entry from '<new>' to 'Exit' (right).*

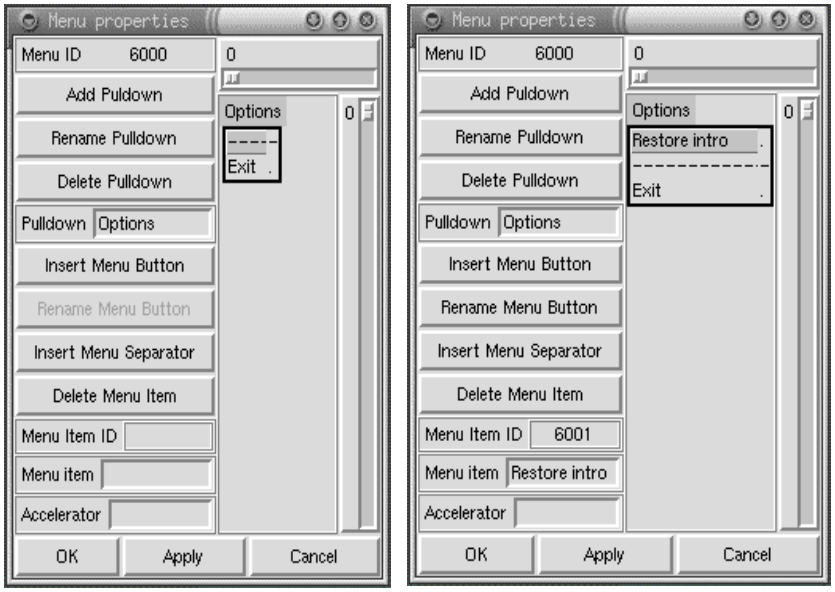


*Figure 23. Addition of a separator object (left) and another menu button object (right) to the Pulldown of the Menu element.*

duplicated in the **Pulldown** and **Menu item** entries respectively. Next, change the contents on the **Pulldown** entry to 'Options' and then press the **Rename Pulldown** button to inact the change on the Pulldown title in the menu properties dialog. Note that the selection on the '<new>' Menuitem object must be maintained for this to work. The center of Figure 22 shows how the altered Pulldown label should appear.

The next change is to rename the Menuitem object labeled '<new>'. Keeping the Menuitem object selected, change the **Menu item** entry to 'Exit' and then press the **Rename Menu Button** button to inact the change. The renamed Menuitem object should appear as in the right side of Figure 22. Next press the **Insert Menu Separator** button to insert a Separator object above the Exit Menuitem object. Note that the selection now switches to the Separator object. The left side of Figure 23 shows how the addition of the Separator object should appear.

Finally, add another Menuitem object above the Separator object on the menu. Keeping the Separator object selected, change the **Menu item** entry to 'Restore intro' and then press the **Insert Menu Button** button to add the new Menuitem object. Note that the selection now switches to the new Menuitem object. The right side of Figure


*Figure 24. Modified Menu element after configuration via the Menu properties dialog.*

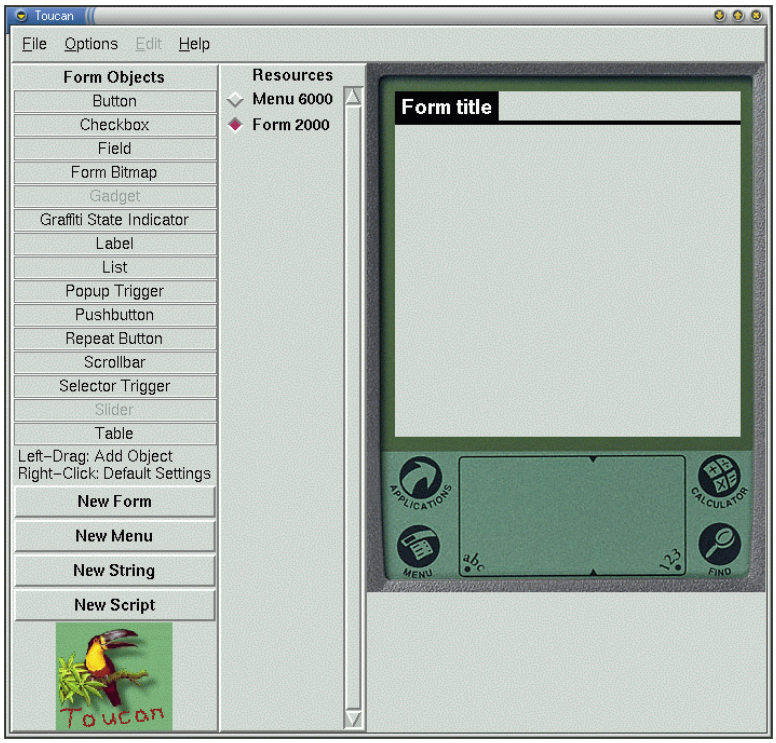


*Figure 25. Initial raw Form element created via the 'New Form' button.*

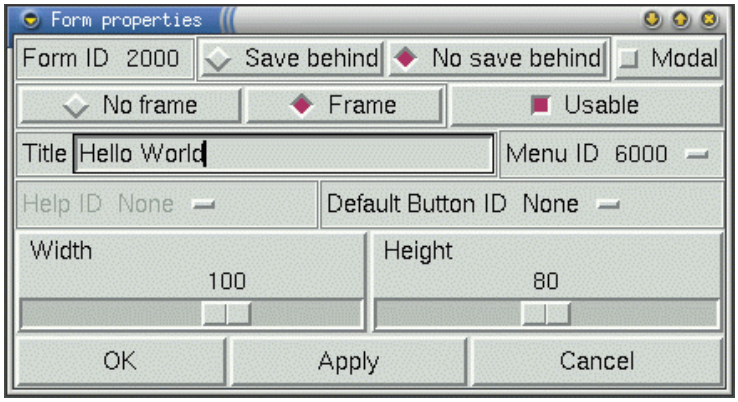


*Figure 26. Setup of Form element properties for the example.*


*Figure 27. Final size and position of the Frame element.*

With the design of the Menu completed, the Form element will now be designed. This is started by pressing the button, **New Form**, to create a new Form element, as shown in Figure 25. Position the mouse cursor over the new Form and double-click the right mouse button. This will display the resource control menu. Click on the **Properties** menu button to display the Form properties dialog. Change the properties so that Form has a **Frame**, has **No save behind** selected, the **Title** is 'Hello World!', the **Width** is 100 pixels, and the **Height** is 80 pixels. Also change the **Menu Id** option from 'None' to '6000' to associate that Menu element created prevously with the Form element. All other properties should be unchanged. The properties should match those in the Form properties dialog window shown in Figure 26. Note that the only value available for the **Menu ID** is 'None'. Press **OK** to commit the changes to the Form properties. Now grab the form by positioning the mouse cursor and pressing and holding down the right mouse button. The coordinate indicator will appear in the upper left-hand corner of the workspace. Move the

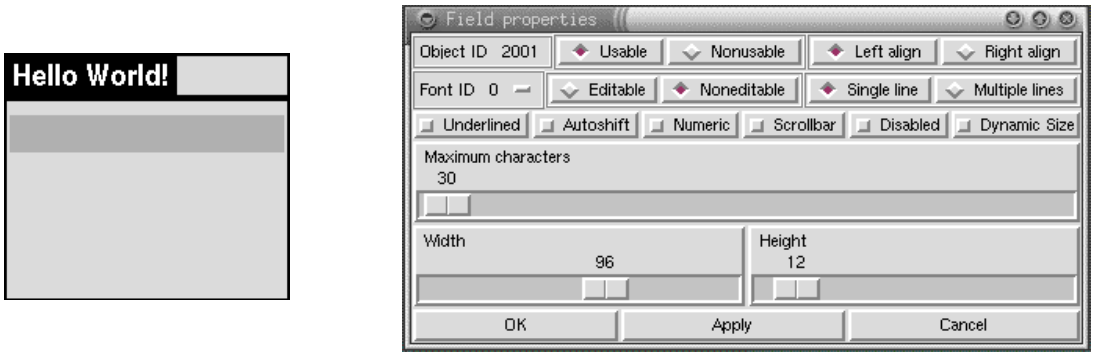


*Figure 28. The Field object placed on the Form element (left) and the Field object properties dialog (right) .*

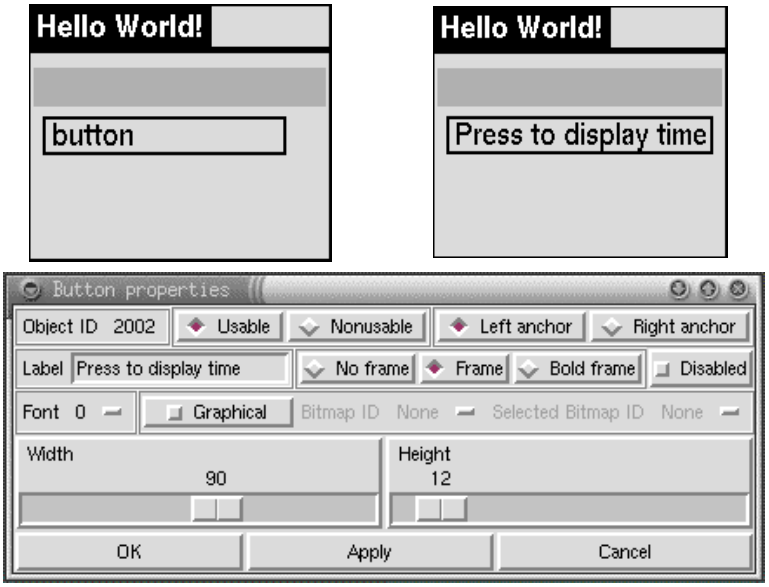


*Figure 29. Initial appearance of the Button object after drop onto the Form element (top left) and its final appearance (top right) after configuration with the Button properties dialog (bottom).*

mouse, keeping the right button depressed until the indicator displays **30,40**, the coordinates of the top, left corner of the form. Release the right mouse button to complete the task. The Form element should now appear as shown in Figure 27.

Next, create a Field object and place it on the Form. Position the mouse cursor onto the Form object type labeled **Field**. Pressing the left mouse button creates an instance of a Field object floating under the mouse cursor. While keeping the mouse button depressed, drag the Field object over to the workspace and onto the Form element. Releasing the mouse button with the mouse cursor over the Form element should place the Field object onto the Form element. If the Field object disappears and an error beep occurs, then the mouse cursor was not properly positioned over the Form element. As with the Form element, position the mouse cursor over the Field object, display the resource control menu, and select the **Properties** menu button to display the Field object properties dialog. Change the properties so that Field is **Noneditable**, has a **Maximum characters** setting of 30, and a **Width** of 96 pixels. All other properties should be unchanged. The properties should match those in the Field properties dialog window shown in bottom part of Figure 28. Press **OK** to commit the changes to the Field object properties. Now reposition the Field object so that its top, left corner is at coordinates **2, 20**. The Field object should now appear as shown in Figure 28.

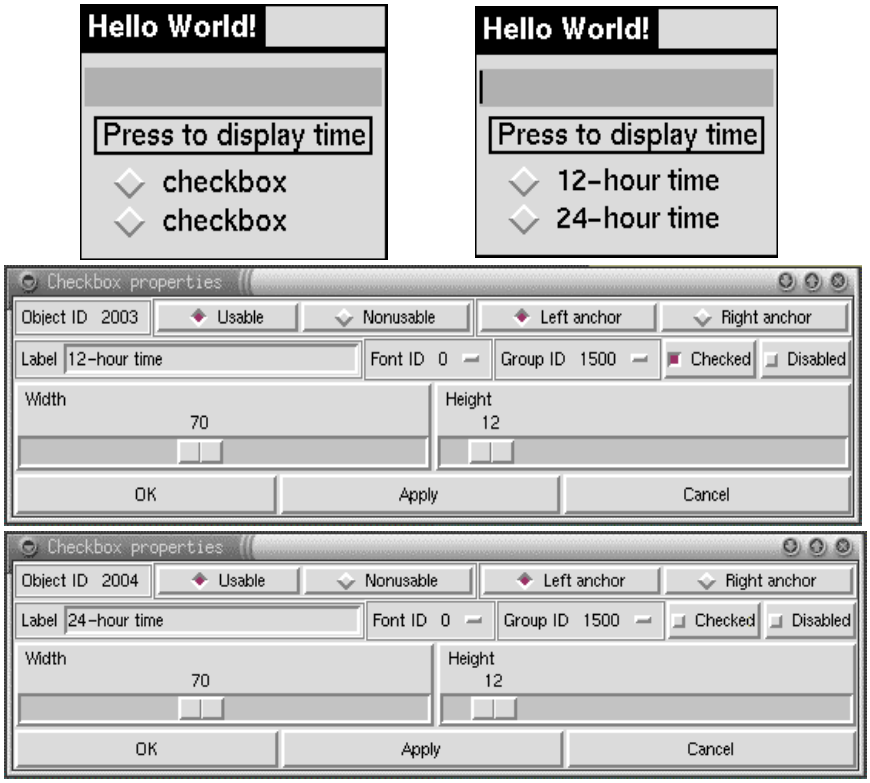


*Figure 30. Initial appearance of the Checkbox objects (top left) and their final appearance (top right) after configuration via the use of the Checkbox properties dialog twice (center and bottom).*

Next, create a Button object in a fashion similar to that of the Field object. Drag a Button object from the **Button** object type label to the Form element on the workspace. Place it below the Field object as shown at the top left of Figure 29. Position the mouse cursor over the Button object, display the resource control menu, and select the **Properties** menu button to display the Button object properties dialog. Change the properties so that the Button has a **Label** of 'Press to display time' and a **Width** of 90 pixels. All other properties should be unchanged. The properties should match those shown at the bottom of Figure 29. Press **OK** to commit the changes and reposition the Button object so that its top, left corner is at coordinates **5, 35**. The Button object should now appear as shown at the top right of Figure 29.

Now, create two Checkbox objects in the same manner as the Button object. The first Checkbox should be placed on the Form element immediately below the Button object. The second Checkbox should be place just below the first Checkbox. They should appear as shown at top left of Figure 30. Change the properties of the first Checkbox so that it has a **Label** of '12-hour time', it is **Checked**, and has a **Width** of 70 pixels. On the **Group ID** option menu, select **New**. A value of '1500' should appear on the **Group ID** menubutton. The properties should match those shown at the center of Figure 30. Commit the changes and reposition the first Checkbox so that its top, left corner is at coordinates **10, 50.** Change the properties of the second Checkbox so that it has a **Label** of '24-hour time' and has a **Width** of 70 pixels. On the **Group ID** option menu, select the ID value of '1500'. The properties should match those shown at the bottom off Figure 30. Commit the changes and reposition the second Checkbox so that its top, left corner is at coordinates **10, 62**. The Checkbox objects should now appear as shown at the top right of Figure 30.

At this point, it would be a good idea to save the work done on this example. A Toucan project file (`.tpj`) can be created by moving the mouse cursor the the menu bar at the top of the main Toucan window and selecting **File/Save.** This is the first time the new project file has been saved, so a Save File dialog window appears allowing the selection of the directory and base name of the Toucan project file. Select a directory in which to save the

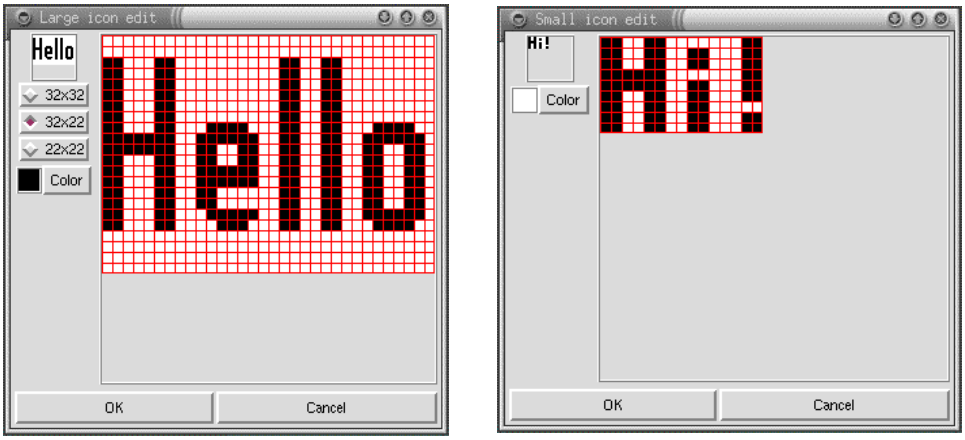


*Figure 31. Design of large (left) and small (right) icons for the example Palm Tcl application.*

project file. The directory needs to be one which you have permissions to write to. In the **File name** field, enter the name 'hello' and press the **Save** button. The Save File dialog window is shown in Figure 5.

## *Setting up application characteristics*

Prior to creating a Palm Tcl application with Toucan, several properties and characteristics need to be set up that are used in the creation of the application and are not necessarily visible from within the application when it runs. These properties and characteristics are found under the Options menu at the top of the main Toucan window. For this simple example (and for most Palm Tcl applications), many of the options and characteristics do not need to be changed from their default settings. Those that need to be changed are described below

Select **Options/Project/Output path...** to display a directory selection dialog window. An example of the dialog window is shown in Figure 7. The directory selected is the destination for all files generated by Toucan and support programs called by Toucan during the process of generating the Palm Tcl resource file. The resource file is the executable program that is loaded and run on the Palm device. Usually the directory chosen is the same as the directory in which the Toucan project file is stored, though it can be any directory the user desires. Select the destination directory by either typing it into the **Selection** entry field or using the mouse to traverse to the desired directory by changing directories through the **Directory** window and pulldown selector. Press the **OK** button to commit the directory selection.

In order for the user to run an application on the Palm device, an icon should appear on the launcher screen. There are two icons that can appear for a given application, the large icon and the small icon. The large icon is displayed when the application launcher preferences are set to view applications as icons. The small icon is displayed when the applications are to be view as an alphabetized list.

Select **Options/Project/Large icon...** to display a dialog window for editing the large icon bitmap. In this dialog window, an image of the 'normal-sized' and 8X normal size 'fat-bit' bitmap images of the large icon are displayed. The user can alter the appearance of the icon by 'painting' pixels in the fat-bit image. Individual pixels can be set to the currently selected drawing color by clicking and releasing the left mouse button while the mouse cursor is positioned over the selected magnified pixel. Multiple pixel may be colored by holding down the left mouse button and dragging the mouse cursor over the pixels to be colored. The change in the icon's appearance is also reflected in the normal-sized image as changes are made. The current drawing color is set by choosing from the available colors in the drop-down menu on the left side of the dialog window. The user can select between one of three standard sizes for the large icon (32x22 is typically used).

Use the mouse to create the large icon as shown in the left of Figure 31. Press **OK** to commit the changes or **Cancel** to abort all changes. Now select **Options/Project/Small icon...** to display a dialog window for editing the small icon bitmap. This window is similar to the dialog window used to edit the large icon. Use the mouse to create the small icon as shown in the right of Figure 31 Press **OK** to commit the changes.

For this example, no changes need to be made under the **Options/PilRC** sub-menu. Select **Options/Par/Application name...** to bring up a dialog window to enter the name of the Palm application. Enter 'Example' into the entry field and press the **OK** button to commit the value and dismiss the dialog. Next, select **Options/Par/Creator ID...** to bring up a dialog window to enter the Palm application's creator ID. Under normal development situations, the creator ID would be a unique four-character value registered with the Palm Creator ID Database at http://dev.palmos.com/creatorid/ (This has been done with the ID used in this example). Enter 'Hi__' into the entry field and press the **OK** button to commit the value and dismiss the dialog. Finally, select **Options/Par/Palm Tcl library path...** to display a directory selection dialog window. The directory selected is the location of the Palm Tcl library file that is combined with other data to form the Palm Tcl resource file that will be loaded onto the Palm device. Select the library directory by either typing it into the **Selection** entry field or using the mouse to traverse to the desired directory by changing directories through the **Directory** window and pulldown selector. Press the **OK** button to commit the directory selection.

At this point, save the properties and characteristics that have been changed for this example. Select **File->Save** again to save the configuration in the Toucan project file. The Save file dialog window will not appear as it did the first time because the project file already exists. Now it is time to create the Palm Tcl script which drives the user interface that has been designed.

## *Creating the Palm Tcl script*

Writing a Palm Tcl script is pretty much like writing any Tcl script, but there are differences. The first difference is that the syntax and command set of Palm Tcl is pretty much that of Tcl 7.6. Tcl 7.6 is a somewhat older, but still very usable version of Tcl. Some of the commands of Tcl 7.6 are not available in Palm Tcl and others are augmented with functions specific to the PalmOS. This needs to be kept in mind for programmers used to newer versions of the language, like Tcl 8.3. The second difference is that there are a number of unique commands in Palm Tcl that are related to supporting the Palm user interface. These commands do not create widgets and manage geometries as with the Tk commands. Instead, the user interface commands in Palm Tcl create handles to preexisting resource elements that are managed by the user to configure and control the objects within the resources. For a full description of these commands, please consult the Palm Tcl: Programming Guide and Reference.

For this example Palm Tcl program, only a few of the commands available in the language will be used. Nevertheless, they are sufficient to create a fully functional Palm program. A top-down approach is taken in the development and explanation on the example Palm Tcl program. Before writing any code, a `STRING` resource needs to be created to hold the script. This is done by pressing the New

The following segment of code initializes the form and displays it on the Palm interface:

```
1. set formHandle [form load 2000 -menucmd {menu %S}]
2. $formHandle itemconfig 2002 -command {GetTime}
3. $formHandle itemconfig 2003 -command {TimeFormat12}
4. $formHandle itemconfig 2004 -command {TimeFormat24}
5. TimeFormat12
6. RestoreIntro
7. $formHandle display
```

Line 1 loads the Form element with resource ID 2000. It also initializes the Menu element assigned to the Form (resource ID 6000) to execute the Tcl procedure `menu` whenever one of its Menuitem objects are selected by the user. The `%S` is an event keyword placeholder that substitutes in the Menuitem resource ID of the Menuitem object pressed. The handle to the Form element is returned and stored in the Tcl variable `formHandle`.

Line 2 accesses the Form element using the command `FORMHANDLE itemconfig`, where `FORMHANDLE` is the value substituted by `$formHandle`. The Button object (resource ID 2002) on the Form element is configured to execute the Tcl procedure `GetTime` whenever the Button object is pressed. Lines 3 and 4 configure the Checkbox

objects with resource IDs 2003 and 2004 to execute the Tcl procedures `TimeFormat12` and `TimeFormat24`, respectively.

Lines 5 and 6 execute the procedures `TimeFormat12` and `RestoreIntro` to initialize a variable and the Field object, respectively. These procedures, along with `menu`, `GetTime`, and `TimeFormat24`, will be explained below. Finally, Line 7 uses the `FORMHANDLE display` command to make the Form element referenced by `$formHandle` visible on the Palm screen. Note that the commands in Lines 1 through 7 are executed in the global scope, so `$formHandle` is a global variable.

The procedure `menu` is executed as an event whenever one of the Menuitem objects in the Menu element (ID 6000) is selected. The procedure is listed as follows:

```
1. proc menu {id} {
2.   switch -exact -- $id {
3.     6001 { RestoreIntro }
4.     6003 { palm launch }
5.   }
6. }
```

Line 1 is the declaration of the procedure. The parameter `id`, the Menuitem's resource ID, is passed to the procedure. The conclusion of the procedure declaration is on Line 6. Lines 2 through 5 are the body of a `switch` operator. The switch option `-exact` indicates that the values supplied by `$id` must match exactly match the pattern before the corresponding branch is executed. All other patterns not matching any of the patters are ignored. If the value matches `6001`, then the procedure `RestoreIntro` is executed. If the value matches `6003`, then the Palm Tcl command `palm launch` is executed. This Palm Tcl command exits the application and invokes the PalmOS application launcher.

The `RestoreIntro` procedure uses the global variable `formHandle` to configure the Field object (resource ID 2001) to contain the string 'Created with Toucan!'. The procedure is listed as follows:

```
1. proc RestoreIntro {} {
2.   global formHandle
3.   $formHandle set 2001 {Created with Toucan!}
4. }
```

Lines 1 and 4 are the declaration of the procedure. Line 2 uses the Tcl `global` command to make the global variable `formHandle` accessible to commands inside the procedure. Line 3 uses the `FORMHANDLE set` command to set the Field object to contain the string.

The procedures `TimeFormat12` and `TimeFormat24` set the global variable `timeFormat` to contain a string used by the Tcl `clock format` command to format a time and date string based upon the current time in seconds. The procedures are listed as follows:

```
1. proc TimeFormat12 {} {
2.   global timeFormat
3.   set timeFormat {%I:%M:%S %p  %m/%d/%y}
4. }
```

```
1. proc TimeFormat24 {} {
2.   global timeFormat
3.   set timeFormat {%H:%M:%S  %m/%d/%y}
4. }
```

Lines 1 and 4 in both procedures are the declaration syntax. Line 2 uses the Tcl `global` command to make the global variable `timeFormat` accessible to commands inside the procedures. Line 3 sets `timeFormat` to the

appropriate format string for either 12-hour time and date or 24-hour time and date, respectively. The `%I` format keyword indicates 12-hour time (1 - 12) for the hours. The `%H` format keyword indicates 24-hour time (00 - 23) for the hours. The `:%M:%S` string contains the minutes and seconds format keywords separated by colons. The `%p` format keyword shows the AM/PM indicator. The `%m/%d/%y` string contains the month (01-12), day (01-31), and two-digit year format keywords separated by slashes.

The `GetTime` procedure uses the global variables `formHandle` and `timeFormat` to configure the Field object (resource ID 2001) to contain a string. That string represents the time and date based on the current value of `timeFormat`. The procedure is listed as follows:

```
1. proc GetTime {} {
2.   global formHandle timeFormat
3.   $formHandle set 2001 [clock format [clock seconds] -format $timeFormat]
4. }
```

Lines 1 and 4 in the procedure are the declaration syntax. Line 2 uses the Tcl `global` command to make the global variables `timeFormat` and `timeFormat` accessible to commands inside the procedures. Line 3 performs three operation. First, the Tcl `clock seconds` command is used to retrieve the current time in seconds from the PalmOS. Second, the `clock format` command takes the current time in seconds and the format string specified by `timeFormat` and returns a string giving the time and date in the appropriate format. Finally, the `FORMHANDLE set` command is used to set the Field object to contain the time and date string.

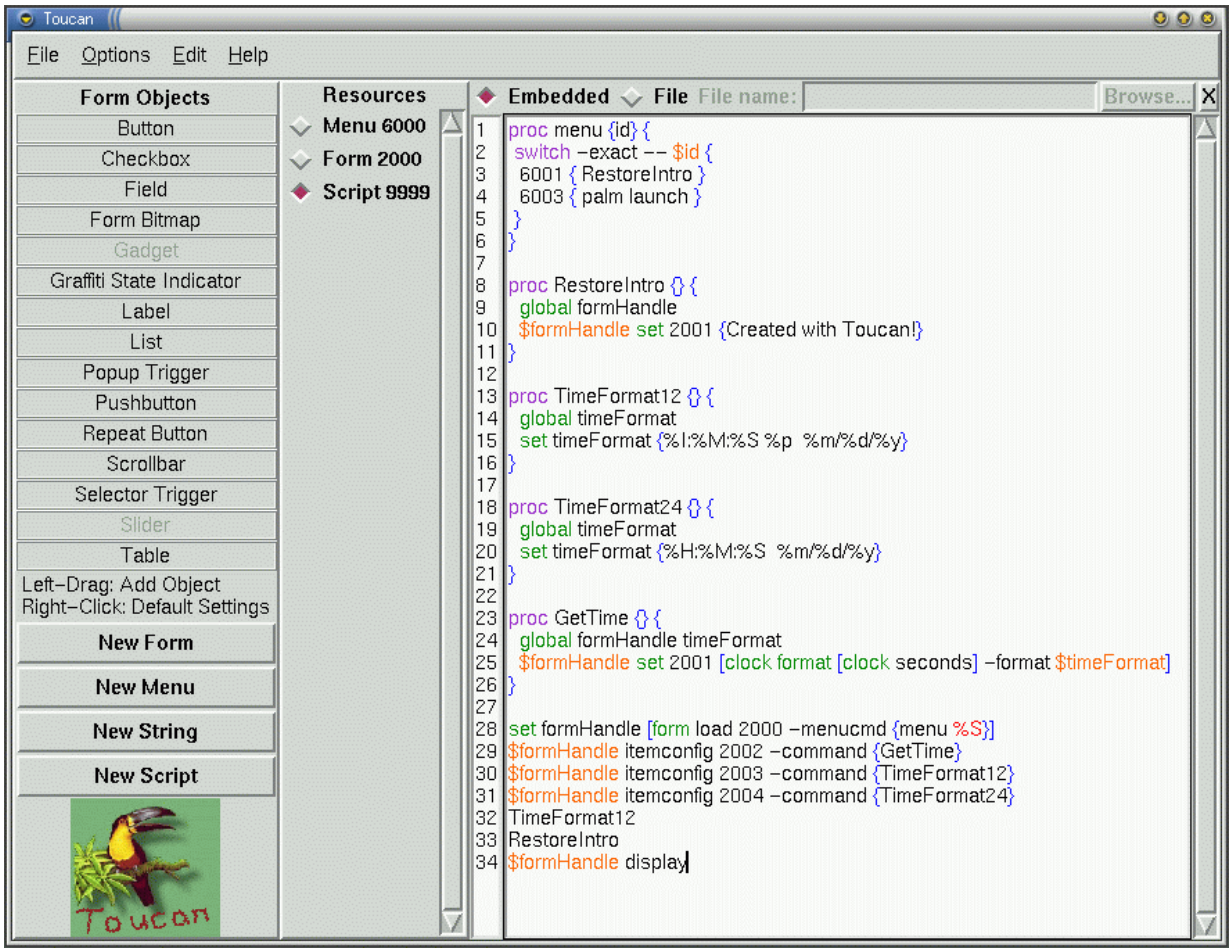

Figure 32. STRING resoruce editor holding the completed Palm Tcl script for the example application.

Combining all of the commands and procedures listed above, the completed Palm Tcl script should appear as shown in the STRING resource editor in Figure 32. Note that the procedures are listed prior to any statemetns that execute them directly in the global scope.

## *Completing, installing, and running the example*

Once all user interface elements are created, application properties and characteristics are configured, and the Palm Tcl script is edited, it is now time to build the example Palm Tcl program. Select **Options/Generate/Target Palm resource file** and answer **OK** to the prompt. The message dialog window should appear similar to that shown in Figure 33. Dismiss this window after the Palm resource build is completed. Below is a listing of what the Palm resource script file, hello.rcp should look like:

```
FORM ID 2000 AT (30 40 100 80)
NOSAVEBEHIND
MENUID 6000
FRAME
USABLE
BEGIN
  TITLE "Hello World!"
  FIELD ID 2001 AT (2 20 96 12) LEFTALIGN NONEDITABLE SINGLELINE USABLE MAXCHARS 30 FONT 0
  BUTTON "Press to display time" ID 2002 AT (5 35 90 12) USABLE LEFTANCHOR FONT 0 FRAME
  CHECKBOX "12-hour time" ID 2003 AT (10 50 70 12) GROUP 1500 USABLE CHECKED FONT 0 LEFTANCHOR
  CHECKBOX "24-hour time" ID 2004 AT (10 62 70 12) GROUP 1500 USABLE FONT 0 LEFTANCHOR
END

MENU ID 6000
BEGIN
  PULLDOWN "Options"
  BEGIN
    MENUITEM "Restore intro" ID 6001
    MENUITEM SEPARATOR
    MENUITEM "Exit" ID 6003
  END
END

VERSION 1 "1.0"
ICON "/home/mcody/wish-scripts/palm/manual/example/hello.bmp"
SMALLICON "/home/mcody/wish-scripts/palm/manual/example/hello_sml.bmp"
STRING ID 9999 "proc menu {id} {\n" \
               " switch -exact -- $id {\n" \
               "  6001 { RestoreIntro }\n" \
               "  6003 { palm launch }\n" \
               " }\n" \
               "}\n" \
               "\n" \
               "proc RestoreIntro {} {\n" \
               "  global formHandle\n" \
               "  $formHandle set 2001 {Created with Toucan!}\n" \
               "}\n" \
               "\n" \
               "proc TimeFormat12 {} {\n" \
               "  global timeFormat\n" \
               "  set timeFormat {%I:%M:%S %p  %m/%d/%y}\n" \
               "}\n" \
               "\n" \
               "proc TimeFormat24 {} {\n" \
               "  global timeFormat\n" \
               "  set timeFormat {%H:%M:%S  %m/%d/%y}\n" \
               "}\n" \
               "\n" \
               "proc GetTime {} {\n" \
               "  global formHandle timeFormat\n" \
```

```
"  $formHandle set 2001 [clock format [clock seconds] -format $timeFormat]\n" \
"}\n" \
"\n" \
"set formHandle [form load 2000 -menucmd {menu %S}]\n" \
"$formHandle itemconfig 2002 -command {GetTime}\n" \
"$formHandle itemconfig 2003 -command {TimeFormat12}\n" \
"$formHandle itemconfig 2004 -command {TimeFormat24}\n" \
"TimeFormat12\n" \
"RestoreIntro\n" \
"$formHandle display\n"
```

Once the Palm resource file, hello.prc is generated, it can be loaded into a Palm device or Pose. Examples of the large icon and small icon are shown on Pose in the left and center parts of Figure 34, respectively. The example application is shown running in Pose in the right part of Figure 34.

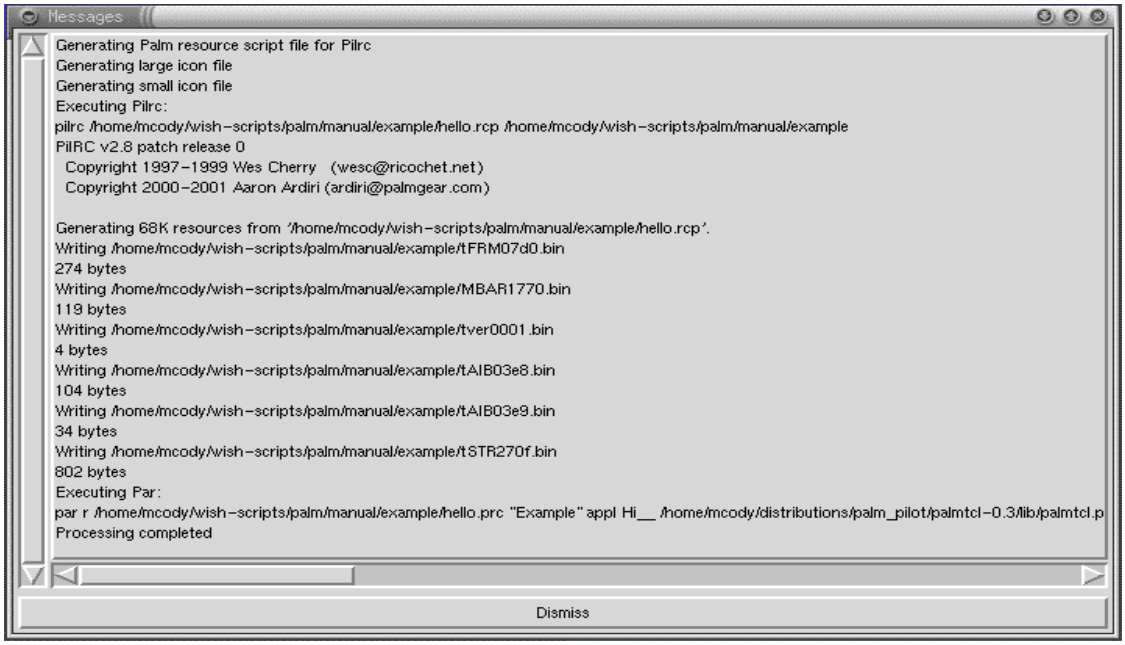


*Figure 33. Messages generated during creation of Palm Tcl resource file.*

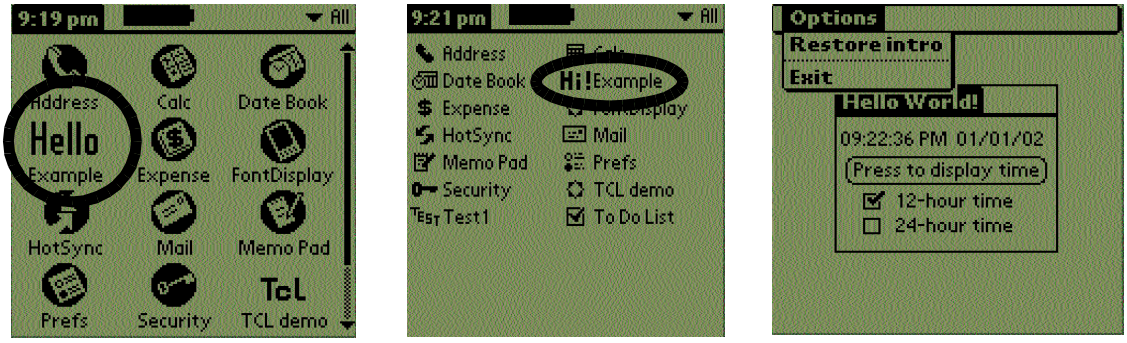


*Figure 34. The example Palm Tcl application running on Pose. The large and small icons are circled in the two Palm launcher views.*

# Appendix A - Version History

## *Change History*

### 1.0 (01-13-2002)

- Initial release.

### 1.1 (03-24-2002)

**BUG FIXES**

- Added missing support for Palm standard font 6 (LED or calculator font).
- Corrected 'catched' file open in toucanLoadProjectFile to work properly with whitespace embedded in path and file names. This enables correct operation in the MS Windows environment where embedded whitespace in file paths is common.
- Corrected title names in some object dialogs so that only the first character was uppercase. This enabled proper keying to default object data arrays.
- Removed the Categories**...** menu entry on the **Project** menu and the **New Alert** button from the main display. Neither function was active.

**ENHANCEMENTS**

- Added local grab to all dialog windows that don't possess it naturally.
- Consolidated Form object properties acceptance code (toucanAcceptProperties).
- Changed **Font ID** entry fields on Form object dialogs to option menus.
- Changed **Menu ID** entry field on Form dialog to an option menu. Menu will dynamically display a selection of all currently existing Menu element IDs.
- Changed **Default Button ID** entry field on Form dialog to an option menu. Menu will dynamically display a selection of all button-like object IDs on the selected Form.
- Changed **Help ID** entry field on Form dialog to an option menu.
- Changed **Bitmap ID** entry field on object dialogs to an option menu.
- Changed **Selected Bitmap ID** entry field on dialogs to an option menu.
- Changed **Group ID** entry field on dialogs to an option menu. Menu will dynamically display a selection of currently existing group IDs. A new group ID can also be generated based upon the lowest available ID number starting with 1500, going to a maximum of 1599.
- Added code to set **Menu ID** references in Form elements to 'None' if the corresponding Menu element is deleted (code added to toucanDeleteObject).
- Added code to set **Default Button ID** references in Form elements to 'None' if the corresponding button-like object is deleted (code added to toucanDeleteObject).

- Added code to manage **Group ID** references in Checkbox and Pushbutton elements as these elements are created, deleted, and assigned to various Group IDs. Internally, Group ID variables are separated from object ID variables.
- Modified width and height scales to allow zero width/height dimension on Button, Checkbox, List, Popuptrigger, Pushbutton, Repeat Button, and Selector trigger object. If the width/height is set to zero, the value output to the Palm resource script (.rcp) file is set to 'AUTO'. The corresponding Palm emulation widgets reflect automatic sizing of the width/height.
- Improved support to the Form element for proper display of modal and non-modal Forms. The **Help ID** is set to 'None' and the option menu is disabled when **Modal** property is turned off. Added code to show a help icon on Modal Forms. Positioning of title and help icon are adjusted according to width of the Form and its modality.
- Added bindings to the **Graffitistateindicator** drag-n-drop start label and the **New Menu** button to sound an error tone when the right mouse button is pressed while the mouse cursor is on them.
- Altered behavior of the **Graphical** option button on the Button, Pushbutton, Popup Trigger, Repeat Button, and Selector Trigger dialogs. If the option is enabled, the **Label** field is cleared, the **Font** selector is set to 'None', and both options are then disabled. The **Bitmap ID** and **Selected Bitmap ID** options are enabled. If the **Graphical** option is disabled, the **Bitmap ID** and **Selected Bitmap ID** options are set to 'None' and both options are disabled. The **Label** field and the **Font** selector are both enabled and the **Font** selector is set to '0'.

**ADDITIONS**

- Support for Bitmap element added. Created dialog window to enable creation, modification, and deletion of Bitmap elements. Selected bitmaps can be displayed via a secondary pop-up dialog. A new Bitmap ID is generated based upon the lowest available ID number starting with 1600, going to a maximum of 1699.
- Added the **Bitmaps...** menu entry into the **Project** cascade menu.
- Support for the Formbitmap object put into place. Drag-n-drop start label enabled and dialog window set up.
- Automatically list all currently available Bitmap elements in dialog windows that contain options supporting them.
- Automatically set resource ID references to 'None' when the Bitmap elements they refer to are deleted.
- Added custom icon to the Toucan main window and dialog windows (MS Windows only).

## 1.2 (08-20-2002)

**BUG FIXES**

- The **Help ID** field in the Form properties dialog was not properly enabled if the **Modal** option was active by default.
- Fixed code to properly toggle on and off the checkbutton or radiobutton widget for a `CHECKBOX` resource object depending upon the state of the **Checked** option.

**ENHANCEMENTS**

- Changed default fonts to larger size.
- Improved the visualization of Modal Form elements. The 'info' icon on a modal Form is displayed on the right-hand side of the titlebar only when a `STRING` resource is selected on the **Help ID** property for the Form. `STRING` resources in the range 1700-1799 can be used by the **Help ID** property.

- The **Project** menu (under the Options menu) has been expaned to include an option to automatically load files containing `STRING` resources that are referenced by the Toucan project. When the project file is loaded, the referenced files will be loaded into separate windows of the MDI editor.
- The **File** menu has been expanded to with entries for loading and saving documents contained in the MDI editor.
- Added prompts to save the current project and unsaved `STRING` resource files will always be displayed prior to exit.

**ADDITIONS**

- Added support for `STRING` resources. The **New String** button is now available for creation of arbitrary string resources . A `STRING` resource ID is generated based upon the lowest available ID number starting with 1700, going to a maximum of 1799. The **New Script** button is creates `STRING` resources for use as Palm Tcl scripts. A `STRING` resource ID is generated based upon the highest available ID number starting with 9999, going to a minimum of 9900.
- Added a built-in, multiple document interface (MDI) editor to support the editing of `STRING` resources. The MDI editor is based on the ctext megawidget (version 2.5.1) developed by George Peter Staplin enhanced to provide electric brace, parenthesis, and quotes matching, and text change detection. When editing Palm Tcl scripts, the editor provides syntax highlighting. The editor content can be stored in an external file (mainitaining compatibility with earlier version of Toucan) or embedded within the Toucan project file and the resource script file processed by *PilRC*. Editor characteristics, such as font type, size, and color, can be changed by the user through a preferences dialog accessed though the **Edit** menu.
- Added standard editing functions for the `STRING` resource editors through both the **Edit** menu and hot keys.

## 1.2.1 (09-08-2002)

**BUG FIXES**

- A bug was discovered regarding the use Toucan with Tcl/Tk 8.4b3. It was discovered that `info patchlevel` yields 8.4b3, which caused problems with `package vsatisfies` and `package vcompare`.. Fixed bugs by regexping `info patchlevel` in pkgIndex.tcl to pull out only a valid version string and replacing `info patchlevel` with `info tclversion` in ctext.tcl. Thanks to Julian Noble for the bug report.
- Fixed a bug in `toucanApplyBitmapData`.. The switch patterns BITMAPGRAY and BITMAPGRAY16 were mis-spelled BITMAPGREY and BITMAPGREY16.
- Fixed a bug in the procedures `toucanGeneratePilrcFile` and `toucanGenerateProjectFile` for both string and scripts. The problem was that double quotes (") and backslashes (\) embedded in strings or scripts were not escaped (\" and \\, respectively) when written to either the resoruce script file (.rcp) or the Toucan project file (.tpj). Thanks to Steve Fehr for the bug report.

**ENHANCEMENTS**

- Reoganized the **Help** menu so it would be placed correctly on the menubar regardless of platform type.

# Index

An index for this manual will be generated at a later time. Sorry if its absence causes any inconvenience for you.

# References

1. Palm Tcl: Programming Guide and Reference, Ashok Nadkarni.
2. PilRC v2.8 User Manual, Aaron Ardiri.