Toucan: A feathered friend for the Palm

Mac Cody

Raytheon Company

Mac_A_Cody@raytheon.com

Abstract

Toucan is a cross-platform, Integrated Development Environment (IDE) for creating applications for Personal Digital Assistants (PDA's) running the Palm OS. The target programming language is Palm Tcl, a port of Tcl 7.6 with extensions to support applications running in the Palm OS environment. Palm Tcl is being developed by Ashok Nadkarni.

The nature of the Palm computing platform and operating system presents several unique challenges to developing applications. Toucan is designed to integrate the tasks required to develop Palm OS applications based on Palm Tcl. Palm Tcl cannot form a working Palm application by itself. It relies upon user-defined Palm resources to create the user interface and static supporting data.

This paper discusses the issues regarding applications development for the Palm OS. Next, the two ports of Tcl to the Palm OS will be presented. Particular attention will be given to Ashok's Palm Tcl package, which uses a clever approach to minimizing the changes to the Tcl 7.6 core. Finally, this paper also discusses how the Toucan IDE streamlines the development of Palm Tcl applications.

1. Introduction

Over the years, Tcl has been ported to a wide range of platforms running a variety of operating systems[1]. Tcl has been successfully compiled to run on Microsoft Windows, numerous UNIX variants (from A/UX to Xenix), and the Apple Macintosh operating system. The port of Tcl 6.7 by Karl Lehenbauer[2] to a handheld computer running an embedded DOS demonstrates how Tcl can be made to fit into very tightly constrained hardware and software architectures and still be able to provide significant functionality to the user.

With the advent of Personal Digital Assistants (PDA's), another class of hardware and software architecture has become fertile ground for new Tcl ports. The current crop of PDA's possess computing performance comparable to a low-end Pentium-class PC or an Apple Macintosh II. This degree of computing performance makes it feasible to port relatively modern versions of Tcl and, possibly, Tk as well.

Two version of Tcl are available for the Windows CE operating system. Akos Polster[3] has ported Tcl 8.3.2 and Rainer Keuchel[4] has ported of Tcl/Tk 8.4a4. Both are in the developmental stage. Brian Webb[5] has ported Tcl 8.3 to the Agenda PDA. The use of Linux and the X Windows System by the Agenda enabled Tcl/Tk to build pretty much "out of the box".

The focus of this paper, though, is on Tcl for the Palm PDA. There are two ports of Tcl to the Palm operating system (Palm OS). Ashok Nadkarni[6] is developing Palm Tcl, which is a port of Tcl 7.6 with extensions supporting Palm OS-specific functions. Almost simultaneously, John Hall[7] released PalmTcl, which is a port based on Tcl 7.4. John has since renamed his port 'Rivendell'[8] to avoid confusion with Ashok's port. The historical details of these ports will be discussed later.

Programming for the Palm OS presents some unique programming challenges. In order to follow Palm's development philosophy, the programmer needs to accept a different mindset regarding the character of an application that will run in the Palm environment. In addition to this, the progamming model for the Palm OS is more constrained and, in some ways, more primitive compared to usual computer programming environments. These issues place additional demands upon the programmer that need to be addressed before an application can be successfully developed for the Palm OS.

Toucan is an integrated development environment (IDE) designed to simplify the creation of Palm OS applications based on Palm Tcl. Toucan is written in Tcl/Tk 8.3.4 and can run on both the Gnu/Linux and Microsoft Windows operating systems. By using Toucan, many of the challenges present in developing for the Palm OS are simplified, if not hidden altogether.

The remainder of this paper will address the issues and challenges related to development for the Palm OS. An overview of the changes made in Tcl 7.6 to create Palm Tcl will also be discussed. The features of the Toucan IDE will be used to illustrate the tasks involved in developing a Palm Tcl application. Finally, some future directions for Palm Tcl and Toucan will be presented.

2. Palm OS development philosophy

The philosophy behind developing applications for the Palm OS is somewhat different than that for regular computers. The elements of this philosophy are based on the restrictions imposed by the hardware of the Palm platform and the way that individuals use their PDA's. The goal is to create applications that follow the credo of 'smaller is better', that take advantage of the Palm's

intuitive interface, and maintain simplicity of use.

The following points highlight the technical issues and design principles that influence the Palm philosophy[9,10]:

- Limited screen size -–Most Palm-based PDA's have a display measuring a modest 160x160 pixels. This limited display real estate requires the developer of an application to strike a balance between providing enough information and overcrowding the screen.
- Application responsiveness - Typically, Palm handhelds are used for brief periods of time, at a moments notice, to enter or retrieve information. A Palm application needs to respond quickly upon launch and throughout its use. Otherwise, the user will quickly become dissatisfied with it.
- A satellite of the desktop PC - The Palm-based PDA is designed to be a portable adjunct to the user's PC. Consequently, PC connectivity is an integral component of the Palm Powered handheld. Many Palm OS applications have a corresponding application on the desktop. The developer may need to address the connectivity issue by developing a conduit to synchronize data between the Palm application and its PC counterpart.
- Limited input methods - The traditional means for users to enter data into the Palm handheld is by using a pen stylus. Data is entered through either writing Graffiti strokes or by using the keyboard dialog provided on the handheld. While external keyboards and other input devices are available for Palm-based PDA's, the developer should not expect their presence for acceptable application performance.
- Low power consumption - Palm-based handhelds use batteries for their power; either AAA alkalines or rechargables. It is not intended that they run an application continuously. Rather, handhelds are turned on long enough to enter or retrieve data and then turned off to conserve battery power. Computationally intensive tasks consume precious battery power and should be implemented in the desktop application instead of the handheld application.
- Limited memory resources - In this age of PC's with hundreds of megabytes of RAM and tens of gigabytes of hard disk storage, Palm-based handhelds have comparatively limited memory resources at both the application level and the programming level. Only within the last year or so have Palm-based PDA's become available with more than 8 megabytes of memory. Many that are still in use have much less than 8 megabytes. The memory must hold both applications and the databases they access. For the application developer, stack space and heap space are severely limited due to the Palm OS's relatively primitive memory management system. Developing for the Palm handheld has a lot of similarities to embedded systems software development. As a result, code optimization is critical. Applications must be written to be as fast and efficient as possible, optimizing for heap space first, speed second, and code size third.
- Non-existent file system - Palm OS does not use a traditional file system. Instead, data is stored in memory chunks called records, which are grouped into databases. An application edits a database in place in memory instead of creating it in RAM and then writing it out to storage.
- Backward Compatibility - Currently, there are an estimated twenty-two million Palm-based devices in circulation. Palm just recently released Palm OS Version 5.0. A large number of Palm handhelds are still using Palm OS versions before Version 4.0. The users of these handhelds either cannot upgrade or do not plan to upgrade their handheld's Palm OS very often. To reach the largest possible user audience, developers needs to design their applications to work with different versions of the Palm OS.

The degree to which these issues and design principles are successfully addressed in an application's design will largely determine that application's acceptance by users. Programming languages, tools, and techniques that aid developers in quickly creating successful Palm Powered applications are rapidly adopted by the developer community.

3. Understanding Palm OS resources

Palm resources make up a significant portion of every Palm application. Acquiring an understanding of what these resources are and how they are used is key to successfully developing Palm applications. Palm resources are primarily static entities of different types that are compiled into a Palm application. Each resource is known to the application and the Palm OS by a unique resource identification (ID) number assigned to it during the development of the application.

Palm resource elements can be categorized into three types. These are operating system support, application support, and user interface. Resource elements which support the operating system contain information that is used by the Palm OS to manage the application once it is installed on the Palm handheld. For example, the **`APPLICATION`** resource specifies the unique, four-character ID that each application must have. The **`ICON`** resource specifies the file of a bitmap that is displayed as an icon for the application in the Palm launcher window. A complete list of these Palm resources is shown in Table 1.

Palm resource elements, which support the application, contain data used during the execution of the program. For example, the **`STRING`** resource contains a character string, or a file referencing a character string, that can be read by the

application. The **MIDI** resource contains data that can be played by the Palm OS MIDI player library. A complete list of these Palm resources is shown in Table 2.

The most visible Palm resources are those that support the application's user interface. A partial listing of these resources is presented in Table 3. Some of these resources describes how data is displayed within the application. For example, the **FONT** resource is used to define custom fonts for the application. The **TRANSLATION** resource is used to provide language translations of text strings in order to support internationalization of the application.

Other user interface resources provide instance and configuration information for various controls available for entering or displaying information. These control resources are in many ways similar to the widgets in the Tk widget library. The **ALERT** resource provides support data for a function that is similar to the *tk_messageBox* function. The data in the resource describes the icon, message, and buttons displayed in a modal alert window. The large variety of bitmap image resources (e.g. **BITMAP**, **BITMAPGREY**, and **BITMAPCOLOR**) came about as Palm handhelds acquired grayscale and color displays. The **FORM** resource is used to create a widget that is comparable to the Tk *toplevel*. Multiple forms of various sizes can be created and displayed in a Palm application. Each form is used as a supporting framework for other user interface objects.

Within each **FORM** resource, other resources are listed, which describe the user interface objects displayed with the form. Most of these objects have counterparts in the Tk widget set. For example, the **FIELD** object is comparable to the Tk *entry* widget, though it can have multiple lines. The **GADGET** object is somewhat like a primitive Tk *canvas* widget. It provides the means for developing custom user interface objects. The Palm datebook calendar displays were developed using the **GADGET** object. The user interface objects that can be created through the **FORM** resource are listed in Table 4.

The **MENU** resource is used to describe a menu bar that can be assigned to a form object and displayed when the Palm handheld's Menu button is pressed. A menu bar consists of one or more menus described by **PULLDOWN** objects. These, in turn, contain one or more **MENUITEM**s. Each **MENUITEM** can be described as either a button or a **SEPARATOR**.

During the design of a Palm application, the developer must be aware of which ID number is assigned to each resource. If a program attempts to use an invalid ID number or if the resource accessed is not of the expected type, unexpected results will occur. Quite often, this will result in crashing the Palm handheld, which is recoverable only through a system reset.

Certain resource IDs are reserved for specific uses in Palm applications. For example, all IDs of value 10000 or greater are reserved for use by the Palm OS. The **VERSION** resource element must have an ID of 1 or 1000 in order to be accepted by the default Palm OS launcher. Though not required, assigning resource IDs of 1000 for the **ICON** resource element and 1001 for the **SMALLICON** resource element ensure that bitmap size checking will be performed on the appropriate object.

Once the resources for an application are defined and placed into a resource script file, they can be compiled into a binary format that can be used in a Palm application. A

APPLICATION	**ICON**
APPLICATIONICONNAME	**ICONFAMILY**
BOOTSCREENFAMILY	**KEYBOARD**
CATEGORIES	**LAUNCHERCATEGORY**
COUNTRYLOCALISATION	**SMALLICON**
	SMALLICONFAMILY
FEATURE	**TRAP**
GRAFFITIINPUTAREA	**VERSION**

Table 1. Palm resources for operating system support. These resources are used by the Palm OS to manage the application. Related resource types are grouped together.

BYTELIST	**MIDI**
DATA	**STRING**
HEX	**STRINGTABLE**
INTEGER	**WORDLIST**
LONGWORDLIST	

Table 2. Palm resources for application support. These resources provide static data for the application.

ALERT	**FONT**
BITMAP	**FONTINDEX**
BITMAPGREY	**FORM**
BITMAPGREY16	**MENU**
BITMAPCOLOR16	**PALETTETABLE**
BITMAPCOLOR	**TRANSLATION**
BITMAPCOLOR16K	
BITMAPCOLOR24K	

BUTTON	**LABEL**	**REPEATBUTTON**
CHECKBOX	**LIST**	**SCROLLBAR**
FIELD	**POPUPLIST**	**SELECTORTRIGGER**
FORMBITMAP	**POPUPTRIGGER**	**SLIDER**
GADGET	**PUSHBUTTON**	**TABLE**
GRAFFITISTATEINDICATOR		**TITLE**

Table 3. Palm resources for user interface support. The bitmap types support different pixel color depths.

Table 4. Resource objects contained within the FORM resource that represent user interface objects displayed within forms..

popular tool to perform this task is the *PilRC* (Pilot Resource Compiler) utility.

4. A Tcl on the Palm

The most common programming language used in Palm application development is C. All of the power (and pitfalls) present in C are available to the developer. Alternative programming languages, such as Java (a language subset), BASIC, CASL, Forth, Lua, Pascal, Scheme, and Smalltalk, are also available. In the spring of 2001, Tcl was added to the list. The events surrounding the introduction of Tcl for the Palm is a case of amazing coincidences.

On February 5, 2001, I approached Karl Lehenbauer with the idea of obtaining the Tiny Tcl sources so I could attempt a port to the Palm OS (I had obtained a Palm III in November of 2000). By April 20, 2001 I had access to the Tiny Tcl sources. On April 28, 2001, John Hall released Rivendell. Up until his announcement, there was no prior knowledge that he was working on the port. On May 9, 2001, Ashok Nadkarni announced Palm Tcl 0.1. Likewise, his efforts were unknown until his announcement. The hard work of others had beaten me to the punch twice!

Upon investigating Rivendell, it was found that it implements a small console, much like the Tcl shell, *tclsh*. The lower half of the screen provides a text field. Buttons are provided for insertion of brackets, braces and double quotes into the text field, invoke evaluation of Tcl scripts entered into the text field, and reset the Palm OS. The upper half of the screen provides another text field that displays the results of the script evaluation. Tcl scripts can also be sourced from Palm memos using a modified *source* command.

Many of the Tcl 7.4 commands are implemented, at least in part. Some Palm OS-specific commands are also provided. The *alert* command supports the display of transient messages through the Palm ALERT resource. The *draw* command provides rudimentary drawing functions (pixels, lines, and rectangles). The beginnings of a scriptable user interface are also present. The *form* command allows the user the create, modify, and destroy Form objects are drawn from a pool of hidden objects in a single Form resource. The user is limited to creating up to thirty-two buttons, text fields, labels, checkbuttons, and lists on a single Form element. Although Rivendell has promise, it appears that development has gone into indefinite haitus.

Ashok Nadkarni's Palm Tcl (currently Version 0.4) is based on Tcl 7.6, rather than a more recent version of Tcl. Ashok made this choice because Tcl 7.6 has a much smaller and simpler source code base compared to Tcl 8.0 or later. Tcl 7.6 does not contain many of the newer capabilities of Tcl, such as the Tcl object model, the bytecode compiler, and namespaces. Still, Tcl 7.6 provides a lot of functionality.

In order to minimize the memory footprint and adapt Tcl tothe Palm environment, a number of Tcl commands

append	array	bgerror	break	catch	concat
continue	error	eval	expr	fileevent	for
foreach	format	global	if	incr	info
join	lappend	lindex	linsert	list	llength
lrange	lreplace	lsearch	lsort	proc	regexp
regsub	rename	return	scan	set	split
string	subst	switch	time	trace	unset
update	uplevel	upvar	while		

Table 5. Unchanged Tcl 7.6 commands.

history	case	interp	package

Table 6. Tcl 7.6 commands removed to save memory.

cd	exec	exit	file	flush	glob
load	open	pwd	seek	tell	

Table 7. Tcl 7.6 commands irrelevant to the Palm.

after	close	eof	fblocked	fconfigure
gets	puts	read	socket	vwait

Table 8. Unimplemented Tcl 7.6 commands.

had their functionality modified, while others have been removed altogether. The majority of the Tcl 7.6 commands are unmodified in Palm Tcl. These are listed in Table 5. The commands in Table 6 are excluded from Palm Tcl in order to save memory. The **`history`** command may be reintroduced later in a memory-optimized form.

A number of Tcl commands perform operations that are only appropriate for a file-based operating system. The Palm OS does not have a file system, so these commands are irrelevant. These commands are listed in Table 7. With Palm Tcl still under development, there are some Tcl commands that still need to be ported to operate within the Palm OS environment. These commands are listed in Table 8. Most of these commands are intended to support socket-based communications. They must be modified to operate within the constraints of the Palm OS's TCP/IP stack.

Three Tcl 7.6 commands have been modified in order to better support the Palm OS. These commands are:

`clock` Removed the **`scan`** subcommand and added the **`selectday`** and **`selecttime`** subcommands.

`source` References a Palm OS resource ID rather than a file name.

`info` Added the **`memstat`**, **`palmtcl_version`**, and **`palmtcl_patchlevel`** subcommands.

Eight new commands are also added to Palm Tcl. These commands support Palm OS-specific functions:

beep Play one of several system sounds.

clipboard Store and retrieve text from the clipboard.

dm Create and manipulate database resources.

emulator Access Palm OS emulator (*pose*) features.

event Programmatic generation of events.

fatal Raise a fatal exception; reset the Palm OS.

form Create and manipulate user interface **FORM** resources.

palm Operations to display the popup keyboard, display the graffiti help, and exit Palm Tcl and invoke the applications launcher.

Porting Tcl 7.6 to the Palm OS is more than just the judicious selection or exclusion of different Tcl commands. The system library functions that make up the Palm OS are not the same as those found on desktop-based operating systems. As a result, some translation needs to be made between the system functions called by Tcl and the system functions that are available on the Palm OS. Ashok developed a compatibility library, aptly called *apnlib*, which provides the translation functions.

Apnlib consists of a number of functions supporting memory management, application debugging, and miscellaneous utility functions. Table 9 lists the *apnlib* functions that bridge between Standard C functions and those provided in the Palm OS. The Palm OS does not provide scanf-style functions. These are implemented directly in *apnlib* from sources derived from OpenBSD.

Apnlib also provides some supporting functions for Tcl 7.6. The Tcl library (libTcl) has, for some time, provided a platform and compiler independent interface for memory allocation. This interface isolates the libTcl library functions from the memory allocation functions of the operating system. Table 10 lists the application interface memory allocation functions within *apnlib* that support the libTcl interfaces and the *apnlib* memory interface functions that they call in turn.

Macros are provided in the *apnlib* headers that define the Standard C and Tcl functions, described above, in terms of the corresponding *apnlib* functions. Substitution is performed by the C preprocessor at compile time. As a result, any source code that uses these functions is automatically portable to the Palm OS platform without any code modifications.

Debugger tracing functions are important as they aid in embedded software development. The Palm OS certainly qualifies as an embedded environment due to its restrictive interface and limited memory resources. Traditional debuggers cannot hope to fit within the memory of a Palm handheld along with the application they would attempt to debug. Instead an external debugger would communcate

Std. C	*Apnlib*	*Palm OS*
malloc	APNappMemAlloc	MemPtrNew
realloc	APNappMemRealloc	MemPtrResize
free	APNappMemFree	MemPtrFree
mktime	CvtTmToDateTime	TimDateTimeToSeconds
qsort	APNqsort	SysQSort
sscanf	APNsscanf	n/a
vscanf	APNvscanf	n/a

Table 9. Standard C function (left column) are replaced by apnlib functions (center column) that translate to Palm OS functions (right column). Note that there are no functions comparable to sscanf and vscanf in the Palm OS. They are implemented within apnlib.

libTcl	*Application I/F*	*Memory I/F*
TclpAlloc	APNappMalloc	APNappMemAlloc
TclpFree	APNappFree	APNappMemFree
TclpRealloc	APNappRealloc	APNappMemRealloc

Table 10. The memory allocation functions of libTcl (left column) are implemented using apnlib's application interface functions (center column), which call apnlib memory interface functions (right column).

Apnlib	*Palm OS*
APNTraceInit	HostTraceInit
APNTraceClose	HostTraceClose
APNTraceText	HostTraceOutputVTL
APNTraceBinary	HostTraceOutputB

Table 11. Debugging functions in apnlib (left column) and the Palm OS functions they call (right column).

with the Palm handheld (actually the Palm OS emulator), receiving execution feedback from the application under test via trace functions. The Palm OS trace functions are exposed by the *apnlib* functions listed in Table 11.

As discussed previously, the philosophy of developing for the Palm handheld is driven by several hardware constraints. Probably the most critical constraint is that of memory. Many Palm handhelds have a total of eight megabytes or RAM that is shared between all applications and databases stored on the device.

Memory usage on the Palm OS can be a difficult problem to resolve, especially when it comes to using the stack. In C, when local (automatic) variables are created within a function, they are allocated from the hardware stack. Unfortunately, the default stack depth allocated for Palm OS applications is only 3.25 kilobytes. The size of the

```
#   define ALLOCATE_SWSTACK(p_, interp_, numbytes_) \
      (p_) = APNmemLifoPushFrame((interp_)->swStack, (numbytes_))
#   define FREE_SWSTACK(interp_) APNmemLifoPopFrame((interp_)->swStack)

int
Tcl_ProcCmd(dummy, interp, argc, argv)
    ClientData dummy;                                   /* Not used. */
    Tcl_Interp *interp;                                 /* Current interpreter. */
    int argc;                                           /* Number of arguments. */
    char **argv;                                        /* Argument strings. */
{
#ifdef SMALL_STACK
    register struct Tcl_ProcCmd_locals {
#endif
    Interp *pInterp;
    Proc *procPtr;
    int argCount;
    int i;
    char **argArray;
    Arg *lastArgPtr;
    Arg *pArg;            /* Initialization not needed, but
                           * prevents compiler warning. */
#ifdef SMALL_STACK
    } *pLocals;
#endif
    int result;

    if (argc != 4) {
            Tcl_AppendResult(interp, "wrong # args: should be \"", argv[0],
                    " name args body\"", (char *) NULL);
            return TCL_ERROR;
    }

#ifdef SMALL_STACK
    ALLOCATE_SWSTACK(pLocals, interp, sizeof(*pLocals));
# define pInterp (pLocals->pInterp)
# define procPtr (pLocals->procPtr)
# define argCount (pLocals->argCount)
# define i (pLocals->i)
# define argArray (pLocals->argArray)
# define lastArgPtr (pLocals->lastArgPtr)
# define pArg (pLocals->pArg)
#endif
.
.
.
#ifdef SMALL_STACK
# undef pInterp
# undef procPtr
# undef argCount
# undef i
# undef argArray
# undef lastArgPtr
# undef pArg

    FREE_SWSTACK(interp);
#endif

    return result;
}
```

Figure 1. Excerpts from the Tcl_ProcCmd function, as ported to Palm Tcl. Usage of the software stack is conditionally compiled, based upon the existence of the SMALL_STACK macro. The ALLOCATE_SWSTACK and FREE_SWSTACK macros, defined in tclInit.h, are replaced by calls to APNmemLifoPushFrame and APNmemLifoPopFrame, respectively.

hardware stack can be enlarged at compile time, but doing so takes additional memory from the dynamic heap.

Choosing the best size for the stack can be problematic. Creating too small a stack risks stack overflow during program execution. Creating too large a stack wastes valuable memory that could otherwise be used by the heap. Due to this problem, C-based programs designed to run on Palm handhelds are written to reduce their dependence upon the hardware stack. In the case of porting an existing C program, like Tcl 7.6, to the Palm, a significant rewrite would be necessary.

To resolve the hardware stack problem while avoiding a significant rewrite of the Tcl 7.6 sources, Ashok implemented a dynamically resizable software stack on the

C program, like Tcl 7.6, to the Palm, a significant rewrite would be necessary.

To resolve the hardware stack problem while avoiding a significant rewrite of the Tcl 7.6 sources, Ashok implemented a dynamically resizable software stack on the heap. He created a set of functions in *apnlib* to support the software stack. These functions are listed in Table 12. The 'Lifo' component of these function names refers to the computer science definition of a stack as being a last-in-first-out (LIFO) queue.

APNmemLifoAlloc	**APNmemLifoClose**
APNmemLifoCreate	**APNmemLifoExpandLast**
APNmemLifoMark	**APNmemLifoMarkedAlloc**
APNmemLifoPopFrame	**APNmemLifoPopMark**
APNmemLifoPushFrame	**APNmemLifoResizeLast**
APNmemLifoShrinkLast	

Table 12. Apnlib software stack management functions. A stack is also known as a last-in-first-out, or LIFO, queue.

The software stack is created by allocating a chunk of memory from the heap and using it to store data that would otherwise be placed on the hardware stack. If the chunk of memory is almost full, another chunk of memory is allocated from the heap to extend the software stack. Conversely, chunks of memory are freed and returned to the heap as data in the software stack is 'popped'.

With a larger 'stack' now available, the Tcl functions were modified to take advantage of it. To minimize code changes, Ashok applied a general rule of thumb for each Tcl function to determine whether the local variables were placed on the hardware or the software stack. If a function required at least twenty bytes of local storage, then the local variables were placed on the software stack. Otherwise, the function was left unchanged and allowed to used the hardware stack.

In order to minimize the code changes in the Tcl functions, Ashok made a clever use of the C preprocessor's **#ifdef** and **#define** macro statements. He used them to turn the declarations of local variables into elements of a register structure unique to that function. An example of this scheme, as applied to the **Tcl_ProcCmd** function, is shown in Figure 1.

Along with the structure's declaration, a structure pointer, called **pLocals**, is created. A call to the function **APNmemLifoPushFrame** (represented by the macro **ALLOCATE_SWSTACK**) is then made to allocate the structure storage from the software stack and place its address in **pLocals**. Afterward, **#define** statements are used to define the function's local variables in terms of **pLocals** pointing to the elements of the allocated structure.

At the end of the function, the macros for the local variables are undefined. Doing this, prevents them from interfering with variables in other functions. Lastly, the allocated structure is removed from the software stack through a call to the function **APNmemLifoPopFrame** (represented by the macro **FREE_SWSTACK**). The macro **SMALL_STACK** is defined when compiling for the Palm OS. All code between any instance of **#ifdef SMALL_STACK** and the corresponding **#endif** is compiled into the Palm Tcl library.

The Palm OS-specific functions, provided in Palm Tcl, are written using the same techniques as with the Tcl 7.6 functions. This includes the use of the software stack, along with the attendant macros, when necessary.

5. Introducing the Toucan IDE

From the previous discussions, it becomes apparent that developing Palm applications with Palm Tcl requires the programmer to create and manage multiple resources and to compose Tcl scripts that interact with them. With all of the issues that need to be addressed in developing a Palm application, this task can become daunting.

As with other complicated software development regimes, an integrated development environment (IDE) can help streamline the process by organizing, automating, or hiding many of the more difficult or mundane tasks. This is the purpose for which Toucan IDE was developed. The following functions are provided within Toucan:

- User interface layout and configuration
- Resource creation and management
- Application icon creation
- Bitmap file browsing and selection
- Script and text string resource creation and editing
- Configuration of supporting utilities
- Palm program resource file generation
- Project management and encapsulation

The Toucan IDE consists of a number of components. Each component supports some aspect of developing applications for the Palm OS. Some components are part of the main Toucan application, while others are run using the Tcl *exec* command. A block diagram of the functional components of Toucan and the data managed by them is shown in Figure 2.

The current version of Toucan (1.2) requires Tcl/Tk 8.3.4. Jan Nijtmans' Img extension[11], version 1.2.4, is used to provide BMP file access for the display and edit of launcher icon bitmaps. George Peter Staplin's ctext megawidget[12], 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

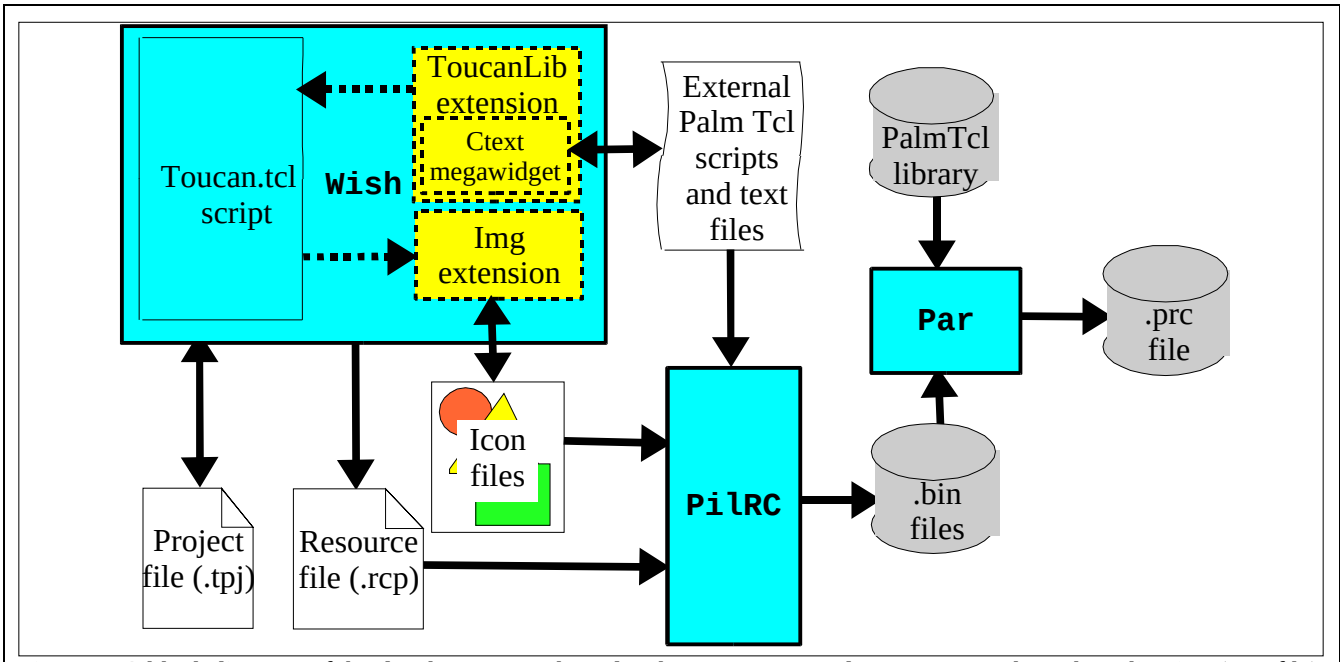

Figure 2. A block diagram of the development tools and Palm resources used to create a Palm Tcl application (.prc file). Toucan integrates various Tcl extensions (yellow boxes) within the windowing shell (wish) along with the command line utilities PilRC and par (blue boxes) . Toucan creates and manages 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.

Palm Tcl scripts. The megawidget is incorporated into the ToucanLib library. ToucanLib is an archive of supporting code modules and bitmaps that can be accessed in a path-independent fashion. That is, the library can be used by Toucan when it is installed into one of the directories listed in the Tcl *auto_path* global variable.

The *PilRC* and *par* command line utilities are used by Toucan to generate the Palm applications. *PilRC* is a program that takes a Palm interface resource script file created by Toucan and generates one or more binary resource files. These can then be combined to create a Palm application. The *par* (Palm database archiver) utility is used to combine the files generated by *PilRC* with the Palm Tcl library file to create a Palm Tcl application. The command-line configurations for *PilRC* and *par* from within Toucan.

The defined resource elements, icon definitions, and utility configurations can be encapsulated within a Toucan project file for later retrieval. A Palm application can then be created over several development sessions. The content of an entire project can be easily moved without concern for misplacing one of the components required to build it.

6. Managing Palm resources within Toucan

Toucan aids the task of managing resources through several means. First, Toucan allows the developer to create **FORM** and **MENU** resources and the resources to be displayed within them graphically. As each resource element is created, it is displayed on a facade that looks like a Palm handheld. This is done using one or more Tk widgets (frame, text, entry, scrollbar, listbox, and various button types) displayed on a canvas widget. The developer can compose the content of **FORM** elements by dragging subordinate resource objects from a 'palette' and dropping them onto the selected **FORM**. The developer can modify the size, position, and content of interface elements until the user interface acquires the desired appearance.

Second, Toucan provides tools to manage **ICON** and **BITMAP** resources. The user can design the launcher icon bitmaps associated with the application. A dialog window is provided to configure and edit the icon bitmaps. Toucan supports creation of both the larger **ICON** resource and **SMALLICON** resource. Editing of the icon patterns is accomplished through a 'fat bit' drawing canvas. Each pixel is represented at eight times its actual size. The **BITMAP** resources are managed via a dialog window. Through the dialog, the developer can create new instances of **BITMAP** resources and the bitmap files associated with them.

Third, Toucan contains multi-document interface editor 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. Each editor window has document change detection to prevent loss of unsaved data.

Fourth, additional resources, related to the identity of the application, are managed through various entries available

within Toucan's Options menu. Resources specifying the application's name, creator ID, and version can be set. Throughout the creation of all these resources, the developer isn't immediately concerned with the resource ID values assigned to the resource elements. As mentioned in Section 3, keeping straight which IDs are assigned to which resources is of crucial importance to successfully designing Palm applications.

Toucan resolves this task 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 number ranges for use by different types of resource elements. As each 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. This technique avoids duplication of resource ID assignments, while maximizing the use of IDs in any given range. Table 13 shows various resource types, the number ranges which are assigned to them, and the increment between ID values for each resource instance.

Resource	*Range*	*Instance Increment*
BITMAP	1600 - 1699	1
FORM	2000 - 5900	100
FORM objects	2x01 - 5x99	1 (up to 99 per **FORM**)
GROUP	1500 - 1599	1
ICON	1000	Only one instance
MENU	6000 - 9900	100
MENUITEM	6x01 - 9x89	1 (up to 6 **PULLDOWN**s, 14 **MENUITEM**s each)
SMALLICON	1001	Only one instance
STRING (arbitrary)	1700 - 1799	1
STRING (script)	9999 - 9900	-1
VERSION	1	Only one instance

Table 13. Resource ID assignments are based upon the type of resource created. FORM objects are any resources that are displayed within a FORM. The GROUP resource refers to PUSHBUTTON or CHECKBOX resources that share the same GROUP ID. STRING resource IDs for Palm Tcl scripts begin at 9999 because the script with that ID is executed first by the Palm Tcl interpreter when it runs.

Figure 3. The Toucan main window displaying the MENU resource element for the example application. The resource control menu is posted on top of the menu bar. From the resource control menu, the ID of the resource can be obtained, the properties dialog can be displayed, or the resource element can be deleted.

7. Developing Palm Tcl apps with Toucan IDE

The following sections describe the creation of a simple Palm Tcl application using the Toucan IDE. This is a four-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 Palm OS. The third step is the creation of the Palm Tcl script which drives the user interface. The fourth step is to invoke the *PilRC* and *par* utilities to generate the executable Palm Tcl application.

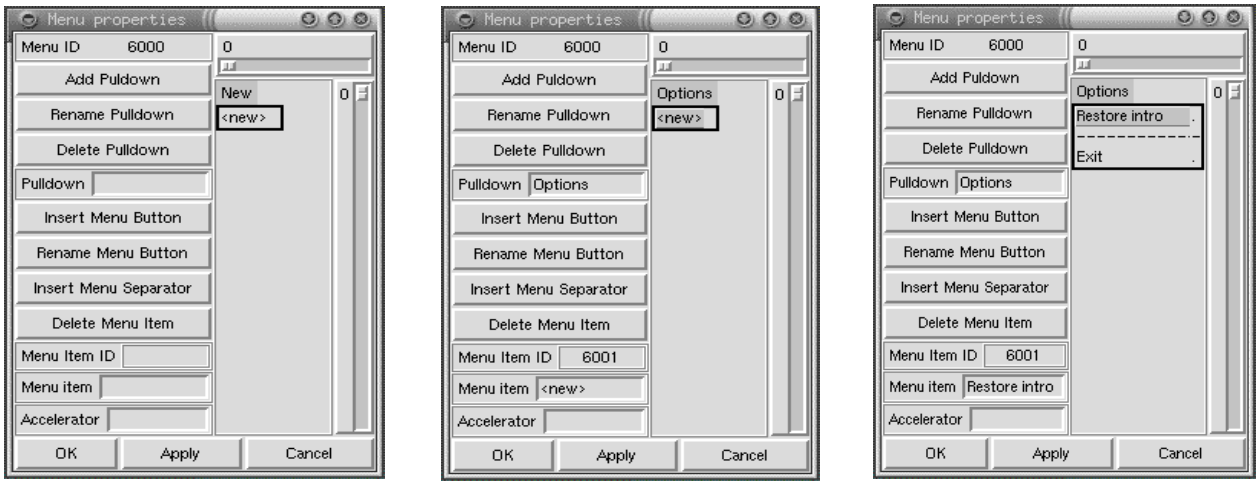

Figure 4. Initial Menu element dialog (left), after renaming a Pulldown entry from 'New' to 'Options' (center) and renaming the original Menuitem entry from '<new>' to 'Exit', and adding a Menuitem SEPARATOR and the Menuitem entry 'Restore Intro' (right).

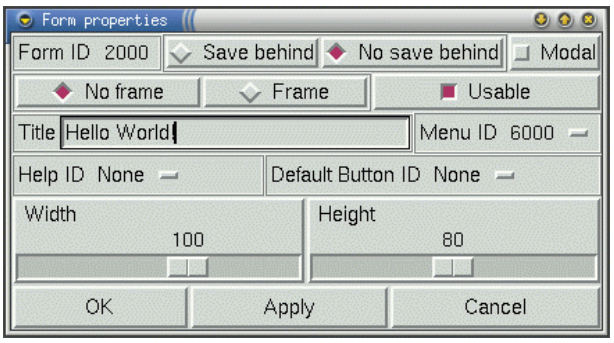

Figure 5. Form element properties dialog window.

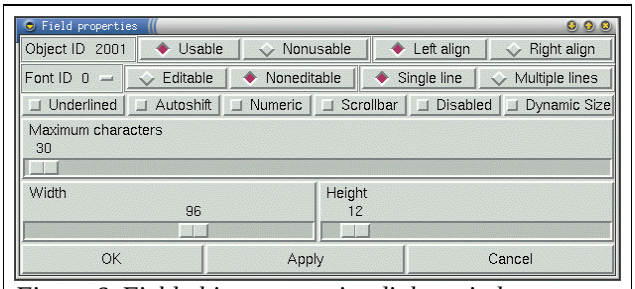

Figure 6. Field object properties dialog window.

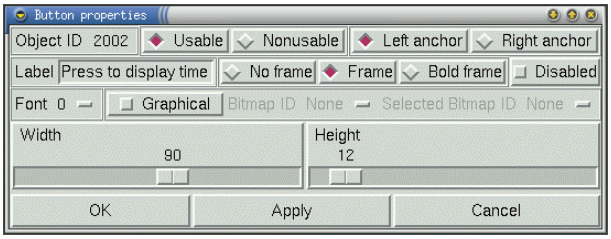

Figure 7. Button object properties dialog window.

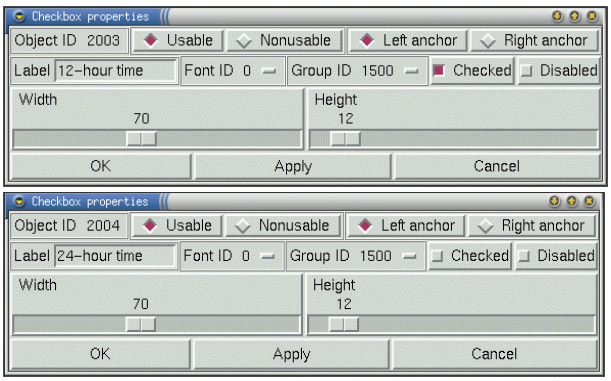

Figure 8. Checkbox object properties dialog windows. The checkboxes share a common GROUP ID, so they behave like Tk radiobuttions.

First, the Menu element will be designed. This is done by pressing the button **New Menu** to create a **`MENU`** resource element, as shown in Figure 3. Note that the **`MENU`** resource is automatically assigned a resource ID of 6000. The right mouse button is double-clicked on the menubar to post the resource control menu. The **Properties** menu button is selected to display the Menu element properties dialog. The initial contents of the Menu properties dialog is shown in the left side of Figure 4.

The Pulldown object labeled 'New' is renamed 'Options'. The altered Pulldown label is shown in Menu properties dialog in the center of Figure 4. The Menuitem button object labeled '<new>' is then renamed 'Exit' and then a second button labeled 'Restore intro' and a Separator object are added above the Exit button. The Menu properties dialog on the right side of Figure 4 shows the changes and additions of the Menuitem objects.

Next, the button, **New Form**, is pressed to create a new **`FORM`** resource element with resource ID of 2000. The resource control menu is then posted and the **Properties** menu button is pressed to display the Form element

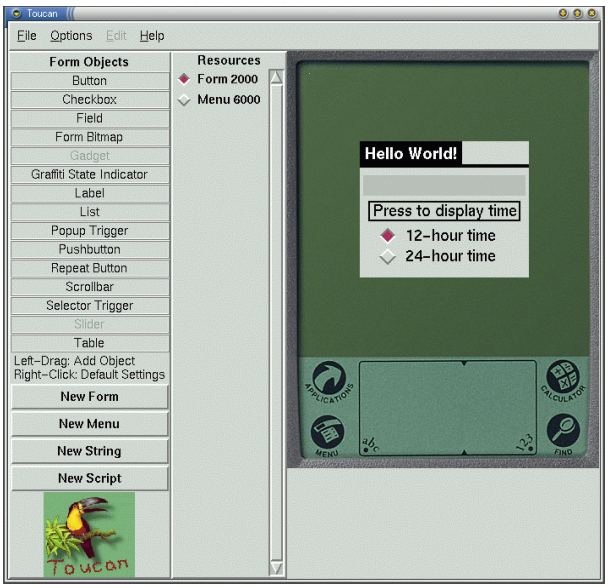

Figure 9. The completed Form element containing Field, Button, and Checkbox objects.

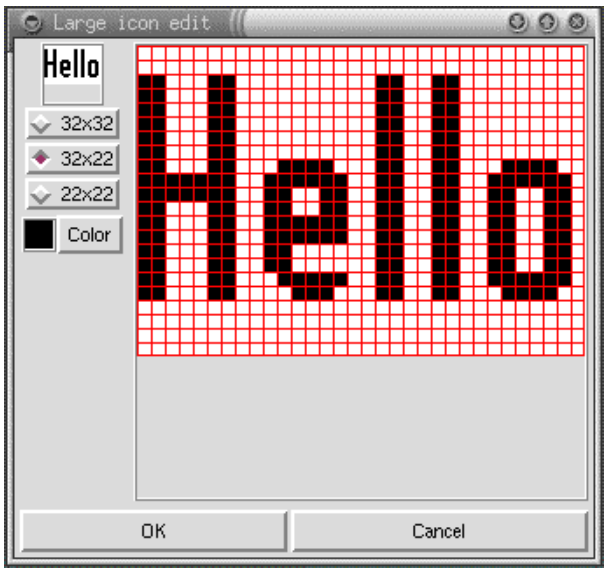

Figure 10. Editing the large icon (ICON) resource with Toucan's built-in icon editor.

properties dialog. The properties are changed so the 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. The **Menu ID** option is changed from 'None' to '6000', to associate the previously created Menu element with the Form element. The Form element dialog widow in Figure 5 shows the new option values. The Form element is repositioned to the middle of the Palm handheld display.

A **`FIELD`** resource object is now placed on the Form. This is done by a drag-and-drop action from the resource object palette on the left of the Toucan main window to the Form element on the Palm facade The Field object properties dialog is displayed and the properties are changed, so that Field is **Noneditable**, has a **Maximum characters** setting of 30, and a **Width** of 96 pixels. The Field object properties dialog widow in Figure 6 shows the new option values. The Field object is repositioned so that it is near the top of the Form element.

A **`BUTTON`** resource object is placed on the Form in a way similar to that of the Field object. The Button object's properties dialog is then displayed. The properties are changed 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 Button object properties dialog widow in Figure 7 shows the new option values. The Button object is repositioned so that it is just below the Field object.

Two Checkbox objects are created in the same manner as the Button object. The first Checkbox is placed on the Form element immediately below the Button object. The second Checkbox is place just below the first Checkbox object. The properties of the first Checkbox are changed so that it has a **Label** of '12-hour time', it is **Checked**, and has a **Width** of 70 pixels. The **Group ID** option menu value is changed from **None** to **New**. A value of '1500' is automatically entered onto the **Group ID** menubutton. The Checkbox object properties dialog window appears as shown in the top of Figure 8. The properties of the second Checkbox object are changed so that it has a **Label** of '24-hour time' and has a **Width** of 70 pixels. On the **Group ID** option menu, the new ID value of '1500' is selected.. With the two Checkbox objects sharing the same GROUP ID, they behave like Tk radio buttons. The properties should match those shown at the bottom of Figure 8. The completed Form element containing the Field, Button, and Checkbox objects is shown in Figure 9.

Now that the user interface is designed, the application's characteristics used by the Palm OS can be set up. In order for the user to run an application on the Palm device, an icon and an application name should appear on the launcher screen. In addition, the application must have a unique creator ID. The creator ID must be unique, because the Palm OS uses it to find and run that application. Palm, Inc. provides a facility to guarantee that a chosen creator ID is unique. The Palm Creator ID Database is located on the Internet at http://dev.palmos.com/creatorid/.

There are two icons that can appear for a given application, the large icon (**`ICON`** resource element) and the small icon (**`SMALLICON`** resource element). 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 viewed as an alphabetized list.

Toucan provides a built-in icon editor in a dialog window to create these two icon resources. In this dialog window, an image of the 'normal-sized' and an 8X normal size 'fat-bit' bitmap image of the large icon are displayed. The

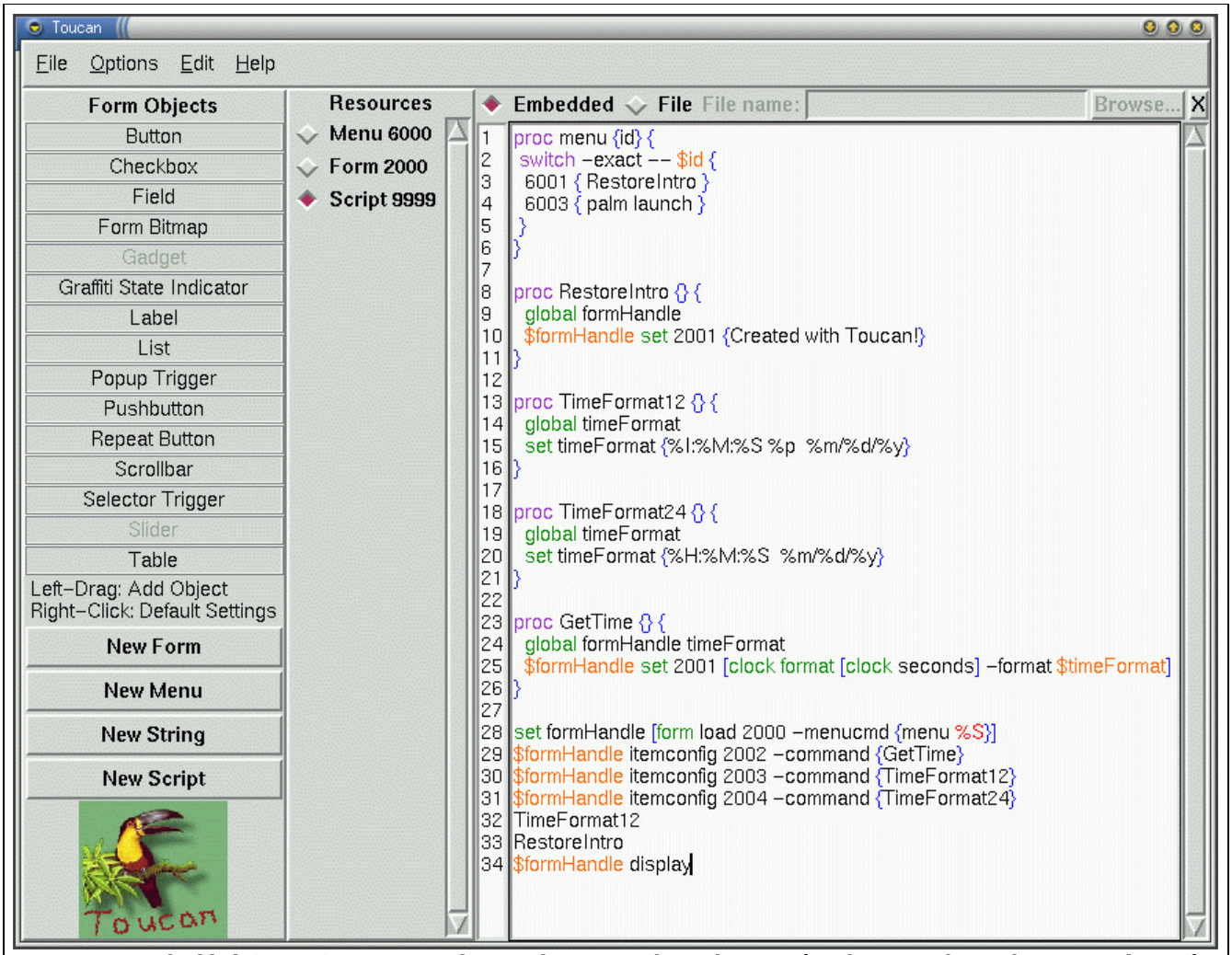

Figure 11. Embedded STRING resource editor. Editing a Palm Tcl script for the example application. Editors for STRING objects that are Palm Tcl script have syntax highlighting.

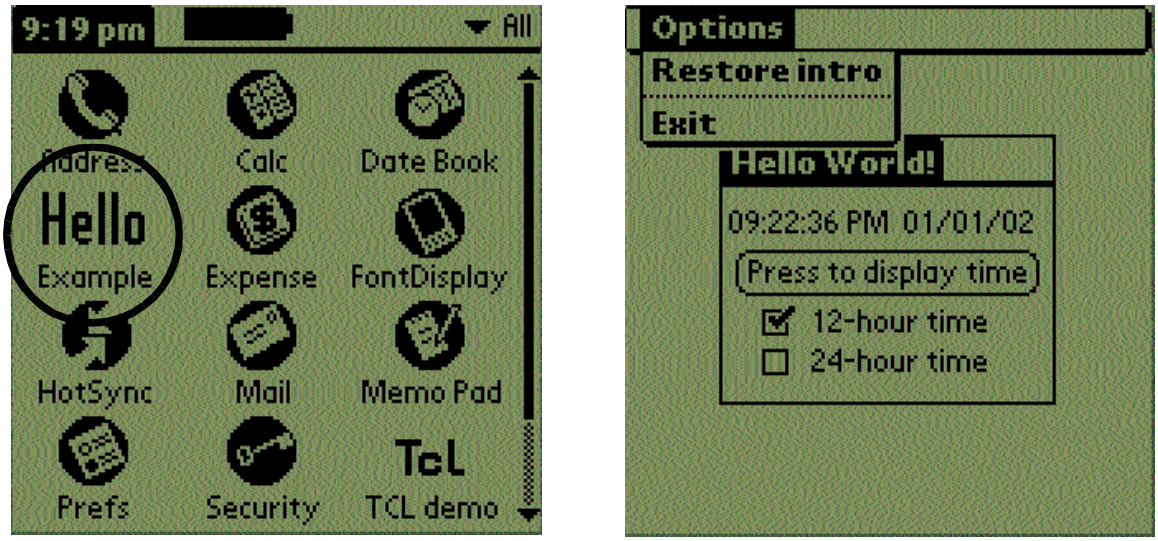

Figure 12. The example application running in the Palm OS emulator. The icon and name the example application is circled on the left. The example application is running on the right.

developer can alter the appearance of the icon by 'painting' pixels in the fat-bit image. Individual pixels can be set in the fat-bit image to the currently selected drawing color (currently either black or white) using the mouse. Changes in the fat-bit image's appearance are also reflected in the normal-sized image. The finished bitmap for the ICON resource is shown in Figure 10.

The application name is specified through Toucan's Options menu under **Options->Par->Application name...** to bring up a dialog window to enter the name of the Palm application. The name 'Example' is entered in this case. Selecting **Options->Par->Creator ID...**, brings up a dialog window to enter the Palm application's creator ID.

The Palm Tcl script which drives the user interface can now be created. The Tcl scripts interact with the Palm OS by accessing the resource elements through their assigned IDs. To develop the Palm Tcl scripts, the developer creates a **`STRING`** resource that will act as a script. The script is entered via the syntax-highlighting editor. Alternately, the script can reside in an external file, which is referenced by the **`STRING`** resource. The script for the example program is shown in Figure 11. Note that resource IDs are used to access the various elements of the user interface. These resource IDs can be quickly obtained by posting the resource control menu for each resource element. Selecting the **ID=<Resource ID>** menu button, places the resource ID into the cut buffer, where it can then be pasted into the script editor.

With the Palm Tcl application designed, the *PilRC* and *par* utilities are invoked to generate the executable Palm Tcl application. The Toucan **Option** menu provides means for setting up the command line parameter for *PilRC* and *par*. These include the location of the Palm Tcl interpreter library, the destination path for generated files, and the target language for the generated files. Just prior to *PilRC* and *par* running, Toucan takes the resource information embodied in the designed application and creates the resource script file. *PilRC* takes the resource script file and generates binary resouce files, which are taken by *par* and combined with the Palm Tcl interpreter library to form the finished Palm Tcl application. In Figure 12, the example application is shown running in the Palm OS emulator (*pose*). The application icon and name are also displayed on the launcher window.

8. Future directions for Palm Tcl

Palm Tcl still has plenty of room for improving its functionality. Many of the resources available on the Palm OS have yet to be tapped. The developer of Palm Tcl, Ashok Nadkarni, has placed me on the development team to assist in the future growth of Palm Tcl.

In the future, support for the following Palm resources and functions will be explored, in no particular order:

- TCP/IP networking
- Infrared communications (IrDA)
- The **`ALERT`**, **`GADGET`**, and **`SLIDER`** resource elements
- Support for drawing
- Expanded Tcl language support

Separate from the future development of Palm Tcl, I'd be interested in exploring the possibilities of porting the upcoming modularized Tcl 8.3.4 core to the Palm OS.

9. Future directions for Toucan

Toucan provides a fairly complete integrated environment for developing Palm Tcl applications. Still there is always room for the improvement of existing and the creation of new support functions. The following list provides a hint to future versions of Toucan IDE:

- Improved user interface: graphical icon palette for user interface components and an icon bar for frequently used operations
- Symbolic reference to resource elements and objects
- An improved icon editor with support for color icons
- Support for different Palm handheld layouts (e.g. Sony Clié has a 320x320 pixel display and AlphaSmart Dana has a 560x160 pixel display)
- Improved look-and-feel of the simulated Palm display
- Syntax checking of Palm Tcl scripts using a modified version of Lindsay Marshall's Frink tool
- Palm Tcl database creation and management
- Integrated Palm application and database up/download
- Integration with *pose* for debugging applications

10. Availability

Toucan IDE version 1.2 can be obtained from my website at http://home.attbi.com/~maccody. I am in the process of setting up a new home for Toucan at SourceForge.net. That website is at http://toucan.sourceforge.net/ and the project site is at http://sourceforge.net/projects/toucan.

11.References

1. 'On What Platforms Does Tcl Run', Tcler's Wiki website, http://mini.net/tcl/2420.html.
2. Lehenbauer, Karl *A Tcl-Powered Handheld Computer for Telecommunications Test Automation*, 8th Annual Tcl/Tk Conference, O'Reilly Open Source Convention, Sheraton San Diego Hotel, San Diego, CA, July 23-27, 2001.