Issue #1: Managing Extensions

John Ousterhout

Computer Science Division
Department of EECS

University of California at Berkeley

Introduction

Goal:

- Make it easy to mix and match various extensions to Tcl and Tk (both C code and Tcl scripts).

Problems:

- Name conflicts.
- Installation is non-uniform and clumsy.
- Proliferation of binaries.

Solutions:

- Naming conventions.
- Installation conventions.
- Dynamic linking, better auto-loading.

Naming Conventions

Problem:

- Each person assumes he/she is the only one building extensions.
- Different packages use same names for global variables and commands, e.g. send.

Possible solution #1: module mechanism

- Tcl provides mechanism for static variables and procedures?
- Still doesn't solve problem for new commands and global procedures.

Solution #2: single command with options

- Like **`string`** command: **`string index`**, etc.
- Still need to find unique command name, unique variable names.

Naming Conventions, cont'd

Solution #3: application prefixes

- For each application or extension, pick a short prefix:
  ```
  expect_
  xp_
  tk_
  dp_
  ```
- Use prefix in all global names (variables, commands, procedures):
  ```
  xp_send
  tk_priv
  dp_rpc
  ```
- Suggestions for uniformity:
  - Only one underscore per name.
  - Use capitalization at internal word boundaries.
- Example: **`tk_menuBar`**, not **`tk_menu_bar`** or **`tk_menubar`**.

Other Naming Issues

Clashes in prefixes?

- Establish registry for prefixes.

Solution #4: object-oriented commands

- Like Tk widgets.
- One command to create object, returns object name: **`button .b`**.
- Use object name as command name, put action as first argument: **`.b invoke`**.
- Avoids command space pollution: only one new command (plus object commands).
- Can provide uniform actions for many different kinds of objects.
- Must allocate unique object names (similar to choosing unique prefix).

Installation

Scripts are easy:

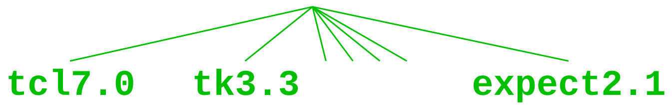
- Put **`.tcl`** files in a directory.
- Create **`tclIndex`** file.
- Add directory to **`auto_path`**.

C code is hard:

- Where to put source code?
- Must compile extensions.
- Must add code to **`wish`** main program by hand.
- Must make new binary.
- Different packages install differently.
- Incompatible versions.

Source Code Management

- Pick directory to hold sources for Tcl, Tk and extensions.
- Each package or application is a subdirectory of this directory:



- Keep version number in directory name, so there can be multiple versions of the same package.
- Use GNU **`autoconfig`** for configuration.
- Create library as well as application (more below).

Incorporating Extensions

In package:

- Define one initialization procedure:
  `Expect_Init`
  `Dp_Init`
- Init proc takes single argument: Tcl interpreter.
- Calls **`Tcl_CreateCommand`** to create new command(s) for package, performs any other initialization for package.

To use package in application:

- Create procedure **`Tcl_AppInit`** that calls all relevant initialization procedures, invokes application’s startup script.
- Link with relevant libraries.
- No need to modify **`main`**: it calls **`Tcl_AppInit;`** Tcl and Tk provide default **`Tcl_AppInit`**.

Dynamic Linking

Goals:

- Avoid proliferation of binaries.
- More flexible: can add new packages dynamically without recompiling.
- Shared libraries save memory.

How?

- New Tcl command:

  `load` *library initProc*

- I will solicit implementations for various systems, include them in Tcl releases.
- Auto-load support (next slide).
- Must resolve differences in how to compile shared libraries for different systems.

Changes to Auto-Loading

Current approach:

- **`tclIndex`** files have fixed format:

```
tk_dialog dialog.tcl
```

procedure name

file to source

- Index files are parsed, not evaluated.

New approach for Tcl 7.0:

- Index files will be evaluated:

```
set auto_index(tk_dialog) \
    "source $dir/dialog.tcl"
```

- Result: 3-4x faster, more flexible.
- Should accommodate TclX style of auto-loading?
- Can invoke **`load`** instead of **`source`** to auto-load shared libraries.

Summary

- Extension builders should conform to conventions.
- Convert non-conformant packages if possible (at next incompatible release?).
- It should become much easier to take advantage of all the contributed packages.