Fullfilling the Promise of `[package unknown]`

Don Porter

http://math.nist.gov/~DPorter/tcltk/

Mathematical & Computational Sciences Division

Information Technology Laboratory

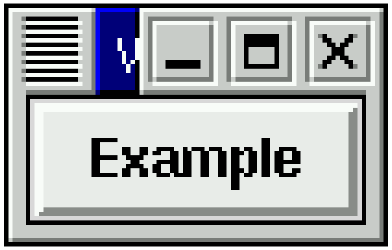
National Institute of Standards and Technology

Gaithersburg, Maryland

Introduction

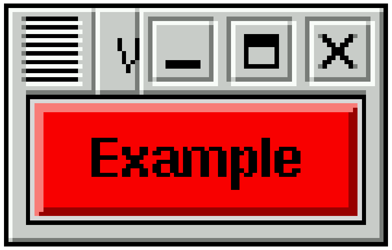
- What color is a Tk button?
  - Naïve answer: Grey.

```
pack [button .b -text Example]
```



  - True answer: Whatever color you want.

```
pack [button .b -text Example -bg red]
```



  - Grey is merely the default.

Introduction

- How does Tcl find and load packages?
  - Naïve answer:
    - Uses `$::auto_path` as a search path...
    - Looks for `pkgIndex.tcl` files...
    - ...created by `pkg_mkIndex`...
    - `[source]`'s them..., etc. ...
  - True answer: However *you* choose!
    - `[tclPkgUnknown]` is merely the default.
    - Customization hook: `[package unknown]`
    - Why don't people know this?
    - Why don't people *use* this?

Outline

- Mini-tutorial on `[package]`
  - Define *provide script* and *index script*
  - Three parties involved in package management
    - Package Author - writes the package
    - Package User - controls the interp
    - Sysadmin - controls the filesystem.
- *de facto* interface: `[tclPkgUnknown]`
  - Features and Limitations
- Propose: New customization hook `[package fsindex]`
  - Sysadmin control over how packages are installed

Tutorial: What is a package?

- A *package* starts as a collection of related Tcl commands, namespaces, and/or variables grouped together as a single unit.
- Grouping simplifies loading and introspection.
- Collection becomes a package by calling `[package provide]`

```
namespace eval ::  {
package provide hw 1.0
namespace eval hw {
 namespace export hello; variable Who world
 proc hello {} {variable Who; puts "Hello, $Who!"}
}}
```

- Load package into interp: eval a *provide script*.

Tutorial: Loading packages

- Evaluation of provide script must call `Tcl_PkgProvide()`
- Simplest possible provide scripts

```
set pscript1 {package provide hw 1.0}
set pscript2 {package provide hw 2.0}
```

- Exactly one version of each package per interp

```
namespace eval :: $pscript1
namespace eval :: $pscript2
conflicting versions provided for package "hw":
1.0, then 2.0
```

Tutorial: Package version selection

- Authors register provide scripts with `[package ifneeded]`

```
package ifneeded hw 1.0 {package provide hw 1.0}
package ifneeded hw 1.1 {package provide hw 1.1}
package ifneeded hw 2.0 {package provide hw 2.0}
```

- Users call `[package require]` to select and eval one registered provide script

```
package require hw 1
1.1
```

- Explains how provide scripts are evaluated.
- How are `package ifneeded` commands evaluated?

Tutorial: Index scripts

- Authors provide an *index script* for each package
- Index script includes `[package ifneeded]`
  - Just as provide script includes `[package provide]`
- Simplest possible index script:

```
set iscript {package ifneeded hw 1.0 {
    package provide hw 1.0}}
```

- User needs evaluation of index script

```
eval $iscript
```

- Mechanism? `[package unknown]` !

Tutorial: `[package unknown]`

- User calls `[package require]`

```
package require $package $requiredVersion
```

- When no registered provide script will satisfy the requirement, `[package require]` evaluates:

```
namespace eval :: [package unknown] [list \
  $package $requiredVersion]
```

- `[package unknown]` returns command to be used for finding and evaluating index scripts.

```
package unknown
tclPkgUnknown
```

Promise of `[package unknown]`

- Why not call `[tclPkgUnknown]` directly?
  - `[package unknown]` enables customization

```
package unknown ?command?
```

- User can select any command for finding and evaluating index scripts.

```
proc falseWitness {p v args} {
 package ifneeded $p $v [list \
  package provide $p $v]
}
package unknown falseWitness
```

- Opens up many possibilities...for the user.

But What About the Sysadmin?

- NOTE: So far no mention of *files*, only scripts
- Left out the sysadmin and the filesystem
- In order to *install* a package, storage in a filesystem must enter the picture.

Provide Script of Installed Package

- An installed package has its provide script in a file.

```
$ cat /usr/lib/hw1.0/hw.tcl
package provide hw 1.0
```

- Scripts registered with `[package ifneeded]` refer to files

```
package ifneeded hw 1.0 \
    {source /usr/lib/hw1.0/hw.tcl}
package ifneeded foo 2.0 \
    {load /usr/lib/foo2.0/../libfoo2.0.so}
```

- But the sysadmin, not the author, decides the installation directory for the package!

Index Script of Installed Package

- The provide script needs to know the installation directory.
  - Unknown to the author who would write provide script.
- Index script has job of registering provide script.
- Give index script job of creating provide script too!

```
package ifneeded hw 1.0 [list \
    source [file join $dir hw.tcl]]
package ifneeded foo 2.0 [list \
    load [file join $dir ..  libfoo2.0.so]]
```

- Interface requirement: index script evaluated in context where `$dir` is the installation directory.

How does `[tclPkgUnknown]` work?

- Called with arguments `$package` and `$requiredVersion`
  - Meant to specify unsatisfied package requirement.
  - `[tclPkgUnknown]` ignores them.
- Treats `$::auto_path` as an input
  - List of directories to be searched
  - Finds files named `pkgIndex.tcl` in those directories and their child directories
- Evaluates the `pkgIndex.tcl` files
  - That's where the index scripts must be.
  - Evaluation in context where `$dir` is directory containing `pkgIndex.tcl`

`[tclPkgUnknown]` Limitations

- Ignores arguments telling what to search for
  - Always evaluates *all* `pkgIndex.tcl` files.
  - Startup time (first `[tclPkgUnknown]` evalutaion)
    - ...not determined by number of packages used.
    - ...not determined by number of packages installed.
    - ...determined by number sibling directories of `[info library]`!
  - Errors in index scripts of all installed packages reported.
    - Even in packages that are never used
  - Keeps no map to be more selective.

[tclPkgUnknown] Limitations

- Shares $::auto_path with the auto-loader
  - Two different methods of loading code into an interp.
  - No reason to assume they should look in the same places.
- Only one pkgIndex.tcl file per directory.
  - Either one package per installation directory, or
  - One index script actually indexes several packages.
- No installed package introspection.
  - [package names] doesn't give what people expect.
- **So why not use [package unknown] to replace [tclPkgUnknown] with something better?**

Unfulfilled Promise of `[package unknown]`

- **Can `[package unknown]` allow sysadmins to customize the way in which packages are installed on their systems?**
  - Not alone.

`[package unknown]`: Half the Battle

- `[package unknown]` only solves half the problem.
  - Enables customized routine for *retrieving* index scripts from the filesystem.
  - User just keeps calling `[package require]`.
- No customizable interface for *storing* index scripts
  - That is, for *installing* packages
  - Authors write installation routines for their packages
    - Follow conventions of `[tclPkgUnknown]`...
    - So do tools like `[pkg_mkIndex]`
  - Once packages are installed as `[tclPkgUnknown]` expects, no other retrieval routine can do better.

The Missing Half...

- Need another customization hook in `[package]`
  - Allow sysadmins to customize package installation conventions on their filesystems.
    - ...like `[package unknown]`
  - Preserve an unchanging interface to authors writing package installation routines.
    - ...like `[package require]`

Proposal: `[package fsindex]`

- The customizable interface to the collection of index scripts stored in the filesystem.
- Customization just like `[package unknown]`

  `package fsindex ?command?`
- `[package fsindex]` with no arguments
  - Returns the command that manages index scripts on the filesystem
- `[package fsindex $command]`
  - Registers `$command` as the command that manages index scripts on the filesystem
- Responsibility of sysadmin to register `[package fsindex]` command matching package installation convention of the filesystem.

`[package fsindex]` Interface

- For each installed package, the command registered with `[package fsindex]` must keep track of
  - ...the package name
  - ...the package version
  - ...the installation directory
  - ...the index script
- Provides access to this information through an interface of several subcommands.

```
namespace eval :: [package fsindex] \
    $subcommand $args
```

- For simplicity, assume `[package fsindex foo]`.

[package fsindex] satisfy

- Implements the `[package unknown]` function.

  ```
  foo satisfy $package $version
  ```

  - Finds all index scripts for all versions of package `$package` that might satisfy the `$version` requirement
  - Limit search to search path set by `[foo searchpath]`
  - Evaluate all found index scripts in search path order, set `$dir`

- New default `[package unknown]`

  ```
  proc default {args} {
    uplevel 1 [package fsindex] satisfy $args
  }
  package unknown default
  ```

[package fsindex] searchpath

- Limit packages searched by instlallation directory.
- Three forms

  `foo searchpath`

  - Returns the current search path

  `foo searchpath set $path`

  - Sets the search path

  `foo searchpath append $directory`

  - Append directory to search path
- Generalizes setting of `$::auto_path`

`[package fsindex] insert`

- Add new index script to the system

  `foo insert $p $v $d $indexScript`

- Part of package installation
- Called after files of version `$v` of package `$p` have been installed into installation directory `$d`
- Registers its `$indexScript`
- Generalization of part of `[pkg_mkIndex]`

`[package fsindex] delete`

- Remove index script from the system

  `foo delete $p $v $d`

- Part of package uninstallation
- Called after files of version `$v` of package `$p` have been removed from installation directory `$d`
- Generalization of deleting `pkgIndex.tcl` file

Increased Promise: Introspection

- What packages are installed, where?

  ```
  foo names

  foo versions $p

  foo directories $p $v
  ```

- Provides what users want from `[package names]`
- Can use with `[foo searchpath]` to carefully limit search to just the directories needed.

Package Providing the Proposed Interface

- Coming Soon: the package `fsidb`
- Simple implementation of the `[package fsindex]` interface
- Collection of index scripts and installation directories associated with packages kept as simple Tcl array read in from a single file.
  - “Poor man’s database”
  - Proof of concept
- “Registry” approach.
  - ...which has its own problems...
- Should mean faster startup times
- Definitely improved introspection

Migration Steps

- Add `[package fsindex]`
- Wrap (parts of) `[package fsindex]` interface around existing `[tclPkgUnknown]`.
- Initialize `[package unknown]` with default that uses `[package fsindex]` to query installed packages.
- Migrate package installation scripts to use new installation interface
  - Installer packages and tools?
- **Still one hook missing to allow use of `fsidb`, etc. ...**

Customized Tcl Initialization for Sysadmins

- Installed packages available for *any* Tcl interp
  - tclsh
  - wish
  - embedded interps
- Index scripts are stored how they're stored.
  - Need to eval matching `[package fsindex foo]` in *all* Tcl interps.
  - Much like `[package unknown tclPkgUnknown]`
- Could "hack" the `init.tcl` file
- ...or `init.tcl` could explicitly eval a system customization file.
  - Feature Request 219375. Patch 403526.

Summary

- A straightforward extension of `[package]` offers
  - ...ability to use other managers of installed packages
  - ...improved introspection
  - ...decoupling from auto-loader
  - ...primitive commands on which to build installers