Eagle: Tcl Implementation in C#

Lang.NET 2009 Symposium - April 16th, 2009

Joe Mistachkin <joe@mistachkin.com>

What is Tcl?

- Tcl (Tool Command Language) is an open-source scripting language created by John Ousterhout in 1988.
- Designed to be highly extensible and easily embeddable.
- Is it relatively “typeless”, has minimal syntax, no fixed grammar, and no keywords.

Who uses Tcl/Tk?

- AOL
- ActiveState
- BAE Systems
- BMW
- BitMover
- Boeing
- Broadcom
- Cisco
- DaimlerChrysler
- EDS
- Eolas Technologies
- F5 Networks
- HP
- IBM
- Intel
- Lucent
- Mentor Graphics
- Microsoft (?)
- Motorola
- NASA (JPL and others)
- NBC
- NIST
- National Instruments
- Northrop Grumman
- Oracle
- Pixar
- Python
- QUALCOMM
- Raytheon
- Sun Microsystems
- Synopsys
- Texas Instruments
- TiVo
- US Department of Defense
- US Postal Service
- ...

What is Eagle?

What is Eagle?

- Eagle (Extensible Adaptable Generalized Logic Engine) is an open-source implementation of the Tcl scripting language written in C# for the CLR.
- Designed to be highly extensible and easily embeddable.
- Is it relatively “typeless”, has minimal syntax, no fixed grammar, and no keywords.

Notable Features

- Dynamic language, loosely coupled with full introspection.
- Uses “duck typing” (like Tcl).
- Supports Unicode (obviously?).
- Integrates with CLR classes.
- Integrates with native libraries.
- Integrates with Tcl/Tk.
- Supports interactive debugging.
- Supports script cancellation.
- Supports read-only variables, commands, etc.
- Supports interpreter-wide variable tracing.
- Supports anonymous procedures and closures.
- Unified unit testing framework.
  - Capable of running tests in Tcl and Eagle.

Notable Features
(cont.)

- Supports custom commands (more on this later).
  - Commands may be added, renamed, or removed at any time and can be implemented in any managed language.
- Supports custom math functions.
  - For use with the expression parser (e.g. the **expr** command).
- Supports binary plugins (i.e. groups of commands).
  - The default plugin provides all the necessary boilerplate code to integrate with Eagle; however, plugins are free to override any default behaviors.
- Supports versioned “packages” which may refer to a binary plugin or a “pure script” package (as in Tcl).
- Supports “hidden” commands.
  - Preliminary support has been added for “safe” interpreter support and custom command execution policies written in managed code.

Syntax: Grouping
(evaluation, step #1)

```
set x 1

puts "grouping with quotes, x = $x"

puts {grouping with braces, x = $x}

proc grouping { args } {
  return [info level [info level]]
}

puts [grouping with brackets, x = $x]
```

Syntax: Substitution

(evaluation, step #2)

```
set name Joe; puts "Hello, $name"

puts [clock format [clock seconds]]

puts "\u2554\u2550\u2557\n\u2551\↩
  ↪\u2551\n\u255A\u2550\u255D"
```

Friendly Errors

```
% set
Error, line 1: wrong # args: should
  be "set varName ?newValue?"

% scope foo
Error, line 1: bad option "foo": must
  be close, create, current, destroy,
  eval, exists, list, open, set,
  unset, or vars
```

CLR Usage Example
(automation)

```
proc threadStart { args } {
   set ::result $args; # do work here...
}

object import System.Threading

set t [object create -alias –parametertypes \
   ParameterizedThreadStart Thread threadStart]

$t Start foo; $t Join; unset t

$::result ToString
```

Native Library Example

(more automation)

```
set x [library declare –functionname \
  GetConsoleWindow -returntype IntPtr \
  -module [library load kernel32.dll]]

set hwnd [library call $x]
```

Tcl/Tk Example
(integration)

```
if {![tcl ready]} then {tcl load}

set interp [tcl create]

tcl eval $interp {eagle debug break}; # type "#go"

tcl unload
```

Custom Commands

- Together with minimal syntax, these are great for creating application-centric domain specific languages (DSL).
- Show example…

Interactive Debugging

- Single-step mode, breakpoints, and variable watches.
- Examine and modify interpreter state.
- Supports isolated evaluation.
- Scripting support via the **debug** command.

Interactive Debugging
(cont.)

- Uses a read-eval-print loop (REPL).
- Debugging “meta-commands” are prefixed with “#” (the Tcl comment character) having special meaning (i.e. they work) only when entered interactively.
- The **#help** meta-command may be used [by itself] to display the list of available meta-commands or with the syntax **#help <name>** to display usage information for a particular meta-command (e.g. **#help #vinfo**).
- Not yet integrated with Visual Studio.

Script Cancellation

(oddly similar to TIP #285, see http://tip.tcl.tk/285)

- Safely cancels a script being evaluated, synchronously or asynchronously.

- Example #1 (from C#):

```
Engine.CancelEvaluate(interpreter,
      true, null, ref result);
```

- Example #2 (from a script):

```
interp cancel -unwind
```

Variable Tracing

- Allows user-defined callback(s) to be executed when a variable is read, written, or deleted.
  - Currently, trace callbacks for a variable can only be specified upon variable creation (via the SetVariableValue or AddVariable APIs).
- Interpreter-wide variable traces are also supported.
  - They can monitor, modify, or cancel all requests to read, write, and delete any variable in the interpreter.

Anonymous Procedures

(compatible with Tcl 8.5)

```
set sum [list [list args] {
   expr [list [join $args +]]
}]

apply $sum 1 2 3; # returns 6
```

Closures

(using **apply** and **scope**)

```
set sum [list [list name args] {
   scope create -open -args $name

   if {![info exists sum]} then {
       set sum 0
   }

   if {[llength $args] > 0} then {
       incr sum [expr [list [join $args +]]]
   }
}]

apply $sum foo 1 2 3; # returns 6
```

Design Philosophy

- Tcl heavily influenced the design of Eagle. In particular:
  - It obeys the “Endekalogue” (i.e. the “11 rules” that make up the Tcl 8.4 syntax).
  - Everything is a string (EIAS).
  - Every command is a string; the first word is the name of the command and the rest are arguments to the command.
  - Commands are free to interpret their arguments however they wish.
  - Every list is a string; however, not every string is a “well-formed” list.
  - The language supplies primitives that can be combined in useful ways.

Design Philosophy
(cont.)

- However, there are some differences in the design that reflect the differences in the underlying platforms:
  - Minimal per-thread data; most state is stored in the interpreter.
    - Interpreters may be used from any thread and/or from multiple threads simultaneously (i.e. they have no thread affinity).
    - Each thread does have its own call stack and current call frame; however, the global call frame is shared between all threads.
  - No interpreter-wide result (i.e. the result of the last evaluation, etc).
    - This merits special attention because it significantly reduces the coupling between components.

What is missing?
(from the Tcl/Tk perspective)

- No Tk commands.
- No argument expansion syntax (introduced in Tcl 8.5).
- No namespace support.
- No asynchronous input/output.
- No slave interpreters, no aliases, and no “Safe Tcl” (explain).
- No **binary**, **fblocked**, **fcopy**, **fileevent**, **format**, **glob**, **history**, **memory**, **scan**, or **trace** commands.
- No **http** or **msgcat** packages.
- No **registry** or **dde** packages.
- Minimal support for the **tcltest** package (just enough to run the test suite).
- For the **open** command, command pipelines and serial ports are not supported.
- For the **exec** command, Unix-style input/output redirection and command pipelines are not supported.

Performance
(from the Tcl/Tk perspective)

- For some operations, Eagle can be up to up to two orders of magnitude slower than “real” Tcl.
- This is to be expected because Tcl is written in highly optimized C, has a mature byte-code compiler, and [most importantly] caches the computed internal representations of lists, integers, etc.

Demonstration

Questions and Answers

Where is it?

http://eagle.to/