Eagle: Maturation and Evolution

Joe Mistachkin <joe@mistachkin.com>

Abstract

Eagle [1] (Extensible Adaptable Generalized Logic Engine) is an implementation of the Tcl [2] scripting language [3] for the Common Language Runtime (CLR) [4].

The initial scope of the project was modest compared to native Tcl; however, it was ambitious considering the sixty day schedule for design, implementation, and testing. Three years have passed since then.

This paper explains how Eagle has matured and evolved since it was presented at the 15th Annual Tcl/Tk Conference [5].

1. Introduction

Eagle has changed significantly since its inception. The primary goal of the project was to provide a complete automation solution for all types of applications running on the CLR. The Tcl language was chosen for its minimalist syntax, simplicity of implementation, and straightforward language semantics.

It has been three years since the start of the project. It has been two years since it was first presented to the public at the 15th Annual Tcl/Tk Conference.

This paper will briefly discuss the major improvements that have been made in terms of performance, ease-of-use, and compatibility.

2. Maturing Code, Evolving Goals

The primary objective for the project, much like for Tcl itself, was to provide a clean, highly extensible library for application automation via scripting. All other goals were secondary, including performance, ease-of-use, and full script-level compatibility with Tcl. The original project schedule was two months. After this initial objective was met, it became clear that the secondary objectives could be achieved incrementally going forward, albeit at a slower pace.

3. Performance

Eagle was not originally designed for raw performance; however, it was designed with performance in mind. Unfortunately, it can be much slower than native Tcl, even for the simplest operations. Profiling reveals that the slowest operations are:

- Parsing strings into lists.
- Building lists from strings.
- Expression evaluation, primarily string-to-type conversions.

All other performance issues are insignificant compared to these three. The primary reason type conversions are so slow is the necessity of starting with the most restrictive numeric type and "failing upward" until something works. If none of the numeric types work, then the operand is a string.

These performance issues were not seen as critical because Eagle was not intended to be used in place of a general purpose programming language. It was intended to "glue" [6] components together while playing nicely in a fully managed code [7] environment and retaining all the dynamic features and expressive power of the core Tcl language.

Subsequently, carefully targeted optimizations to the critical code paths have resulted in significantly better performance today than when Eagle was originally presented. Further research into improving the performance of Eagle is ongoing.

4. Compatibility

The intent of the project was to eventually provide full script-level compatibility with Tcl 8.4 [8]; however, given how aggressive the original project schedule was, certain features had to be omitted from the original release. At the time of this writing, Eagle is still not 100% compatible with Tcl 8.4. This table compares the major missing features between the 15th Annual Tcl/Tk Conference and today:

October 2008	October 2010
No Tk functionality.	No change.
No argument expansion syntax.	No change.
No `namespace` support except the global namespace.	The command and variable resolvers are now extensible; however, namespace semantics are still not implemented.
No asynchronous input/output.	No change.
No server sockets.	Fully implemented.
No `registry` or `dde` commands.	No change.
No `pkg::create` or `pkg_mkIndex` commands.	No change.
No slave interpreters, no hidden commands, no aliases, and no Safe Tcl.	Slave interpreters, hidden commands, aliases, and an extensible policy mechanism have been implemented. The Safe Base is still not implemented.

No Tcl library, such as the `http`, `msgcat`, and `tcltest` packages.	No change.
The following Tcl commands are not implemented: `fblocked`, `fileevent`, `fcopy`, `glob`, `memory`, `binary`, `scan`, `format`, `trace`, and `history`.	The `fcopy` command has been implemented with the exception of the "`-command`" option.
The Tcl library routines [9] matching the patterns `auto_*` and `tcl_*` are not implemented.	No change.
For the `open` command, command pipelines and serial ports are not supported.	No change.
For the `exec` command, Unix-style input/output redirection and command pipelines are not supported.	No change.
For the `load` and `unload` commands, the semantics are not identical to those of Tcl because the binary package management is radically different due to the nature of the CLR.	No change.
For the `clock format` command, some of the Tcl formatting characters are not supported.	All of the Tcl formatting characters from Tcl 8.4 are now supported.
For the `clock scan` command, recognition of relative date/time strings (e.g. "1 day ago", "next Wednesday", etc.) is not supported.	No change.
For the `fconfigure` command, all options except "`-encoding`" and "`-translation`" are not supported.	No change.
For the `regexp` command, the "`-about`" option is not supported.	No change.
For the `regexp` command, using the "`-start`" option with the beginning of a line anchor does not work properly due to lack of support for something like TCL_REG_NOTBOL in the CLR regular expression engine.	No change.
For the `pid` command, supplying the optional *channelId* argument is not supported (always returns an empty string).	No change.
For the `proc` command, arguments	The `proc` command has full support

with default values are not supported.	for arguments with default values.
For the **exit** command, by default does not exit the process; it merely prevents further trips through the engine and the interactive loop.	This is by design. The "**-force**" option can be used to mimic the Tcl behavior.
For the **after idle** command, we evaluate the script immediately prior to the next evaluation because we have no idle detection.	No change.
The **array statistics** command is not supported.	No change.
For the **array** command, the search sub-commands (i.e. "**anymore**", "**nextelement**", "**donesearch**", "**startsearch**") are not supported.	Fully implemented.
For the "*env*" array, the **array names**, **array get**, and **info exists** commands are not supported.	Fully implemented.
Documentation of the integration and extensibility API is incomplete.	No change; however, some progress has been made on tooling for the generation of documentation.
Support for tcltest functionality [10] is incomplete.	The "**test**" command has been fully implemented.
Unit tests are incomplete.	There are now a large number of unit tests. More tests and better code coverage are still needed.

Many people ask why support for namespaces has not been implemented. Originally, it was due to lack of time in the project schedule. Today, it is partially due to the potentially disruptive effects it could have on stability and performance. Another important thing to remember is that namespaces were added to Tcl as a stopgap measure because the number of global variables and procedures in large Tcl applications was becoming unmanageable [11]. Since Eagle is not designed for building stand-alone applications, this should be less of an issue.

It was thought that choosing the CLR as the platform would eliminate or greatly reduce the need for any future porting efforts. This has proven to be incorrect. There are a huge number of subtly incompatible versions, variations, and subsets of the CLR. Each of these is really a distinct platform. The following environments are currently supported:

- Microsoft .NET Framework 2.0, 3.0, 3.5, and 4.0 [12]
- Mono on Windows and Unix [13]

The following are untested:

- DotGNU [14]

The following are not supported:

- Microsoft .NET Compact Framework 2.0 and 3.5 [15]
- Microsoft Silverlight [16]
- Moonlight [17]

5. Refactoring

Much refactoring work has been performed on the code [18]. These efforts have focused on reducing the total amount of code and making it cleaner, more portable, more extensible, and easier to understand. Examples include:

- More consistent naming of variables, methods, and classes.
- Simplified command and plugin integration.
- Simplified host creation and integration.
- More extensible core marshaller.
- Targeted optimizations for all critical code paths.
- More robust threading model.
- Test framework that works with both Tcl and Eagle.
- Better Tcl/Tk integration.

6. The Future

In the near future, some work needs to be done to advance the project beyond the beta [19] stage. The currently planned exit criteria for the beta are:

- No unresolved high-priority bugs.
- At least one production deployment.
- Full script-level compatibility with Tcl 8.4 except the `namespace` and `clock scan` commands.
- Full documentation of the managed integration API.
- Full documentation of the differences between Tcl and Eagle.

7. Acknowledgements

This year, the author wishes to thank Dawson Cowals [20] and Stuart Cassoff [21] for their help.

8. References

[1] Eagle, http://eagle.to/
[2] Tcl Developer Xchange, http://www.tcl.tk/
[3] "Scripting language", http://en.wikipedia.org/wiki/Scripting_language
[4] "Common Language Runtime", http://en.wikipedia.org/wiki/Common_Language_Runtime
[5] 15th Annual Tcl/Tk Conference, http://www.tcl.tk/community/tcl2008/info.html
[6] Glue language, http://en.wikipedia.org/wiki/Glue_language
[7] Managed code, http://en.wikipedia.org/wiki/Managed_code
[8] Tcl 8.4, http://www.tcl.tk/man/tcl8.4/
[9] Tcl library routines, http://www.tcl.tk/man/tcl8.4/TclCmd/library.htm
[10] tcltest functionality, http://www.tcl.tk/man/tcl8.4/TclCmd/tcltest.htm
[11] Namespaces, http://www.beedub.com/book/2nd/Name.doc.html
[12] Microsoft .NET Framework, http://en.wikipedia.org/wiki/.NET_Framework
[13] Mono, http://en.wikipedia.org/wiki/Mono_(software)
[14] DotGNU, http://en.wikipedia.org/wiki/DotGNU
[15] Microsoft .NET Compact Framework, http://en.wikipedia.org/wiki/.NET_Compact_Framework
[16] Microsoft Silverlight, http://en.wikipedia.org/wiki/Microsoft_Silverlight
[17] Moonlight, http://en.wikipedia.org/wiki/Moonlight_(runtime)
[18] Eagle ChangeLog, http://eagle.to/ChangeLog
[19] Beta, http://en.wikipedia.org/wiki/Software_release_life_cycle#Beta
[20] Dawson Cowals, http://www.dawsoncowals.com/
[21] Stuart Cassoff, http://wiki.tcl.tk/10628