Eagle: Tcl Implementation in C#

Joe Mistachkin <joe@mistachkin.com>

1. Abstract

Eagle [1], Extensible Adaptable Generalized Logic Engine, is an implementation of the Tcl [2] scripting language for the Microsoft Common Language Runtime (CLR) [3]. It is designed to be a universal scripting solution for any CLR based language, and is written completely in C# [4]. Superficially, it is similar to Jacl [5], but it was written from scratch based on the design and implementation of Tcl 8.4 [6]. It provides most of the functionality of the Tcl 8.4 interpreter while borrowing selected features from Tcl 8.5 [7] and the upcoming Tcl 8.6 [8] in addition to adding entirely new features.

This paper explains how Eagle adds value to both Tcl/Tk and CLR-based applications and how it differs from other "dynamic languages" hosted by the CLR and its cousin, the Microsoft Dynamic Language Runtime (DLR) [9]. It then describes how to use, integrate with, and extend Eagle effectively. It also covers some important implementation details and the overall design philosophy behind them.

2. Introduction

This paper presents Eagle, which is an open-source [10] implementation of Tcl for the Microsoft CLR written entirely in C#. The goal of this project was to create a dynamic scripting language that could be used to automate any host application running on the CLR.

3. Rationale and Motivation

Tcl makes it relatively easy to script applications written in C [11] and/or C++ [12] and can also script applications written in many other languages (e.g. FORTRAN, Prolog, SML, etc.) However, applications written for a managed execution environment, such as a Java Virtual Machine (JVM) [13] or the CLR cannot easily integrate with native code. In particular, the native call interfaces for Java [14] and the CLR, respectively the Java Native Interface (JNI) [15] and Platform Invoke (P/Invoke) [16], are very complex and somewhat fragile. In the case of integrating with the CLR the normal Tcl implementation can be used, but doing so requires extensive use of P/Invoke, can be very difficult to implement correctly, and does not make full use of the other facilities provided by the CLR.

Until now, there were several approaches to scripting CLR applications. One approach involved using Visual Studio Tools for Applications (VSTA) [17] (the successor to Visual Basic for Applications (VBA)) [18]. Another involved the use of Domain Specific Languages (DSL) [21].

The VSTA approach assumes that the users that wish to customize or script the application have extensive knowledge of programming in one of the supported languages (e.g. Visual Basic.NET [19] or C#). Application customization is accomplished exclusively through managed assemblies, not source code. This approach does not lend itself to using dynamic languages and makes it more difficult to write, test, and debug customizations. It also makes it more difficult to ship customizations for modification and use by others. Finally, redistribution of VSTA can be expensive and requires a per-seat or royalty based license; royalty payments range from one to five percent of net receipts [20].

While the DSL approach may be well suited for some problem domains it is not a general purpose scripting solution.

A third approach involves using one of the "dynamic languages", i.e. languages based on the DLR: examples include IronPython [23] (managed Python [24]), IronRuby [25] (managed Ruby [26]), IronLisp [27] (managed Lisp [28]), IronScheme [29] (managed Scheme [30]), Nua [31] (managed Lua [32]), or Managed JScript [33]. This is essentially a variant on the second approach.

In general, the DLR approach is workable; however, the existing languages based on it are not application centric. They are general purpose programming languages. Eagle fits in this same space. Like Tcl before it, Eagle was designed primarily with the goals of application integration, extensibility, and automation in mind.

4. Original Requirements

The functional requirements included:

- It must be *general-purpose*.
- It must be *thread-safe*.
- It must be *customizable*.
- It must be *localizable*.
- It must be *user-friendly* enough for non-programmers to use.
- It must be *powerful* enough for programmers to use.
- It must be simple to *extend* and *integrate* with.
- It must be capable of *debugging* scripts.
- It must be *exception-safe*.
- It must use the "*strong guarantee*" whenever possible.
- It must assume that all *third-party* code can throw *exceptions*.
- It must *not* let any *exceptions propagate* outward from its public classes and interfaces.

The non-functional requirements included:

- It must be written for the CLR in a managed language.
- It must be suitable for use in all types of CLR based applications (e.g. client, server, console, WinForms, WPF [22], ASP.NET, etc).

Based on the requirements, it was decided that the Tcl language (i.e. the syntax and the core command set) was a good basis for the new scripting language. The following additional requirements were added specifically to address issues related to implementing a Tcl-style dynamic scripting language on the CLR:

- The language syntax must be 100% compatible with Tcl 8.4 (i.e. it must obey the "Endekalogue" [46] or "11 rules").
- The semantics of the core command set should be as backward compatible as possible with those in Tcl 8.4 and as forward compatible as possible with those in Tcl 8.5 and Tcl 8.6.

To ensure a high degree of compatibility with Tcl, several algorithms were blatantly stolen from it:

- The string-matching (i.e. glob style matching) algorithm.
- The script-parsing algorithm.
- The list building and parsing algorithms.
- The script evaluation and substitution algorithms.
- The recursive descent expression parser algorithm.
- Parts of the algorithm for expression evaluation.
- The algorithms for quite a few of the built-in commands (e.g. **`if`**, **`for`**, **`foreach`**, etc).

The script evaluation and substitution algorithms have since been heavily customized while retaining compatibility with Tcl. The expression parser has also been customized, with the inclusion of some additional operators.

Due to the lack of a byte-code compiler, I execute the expression tokens directly, which is a dramatic departure from modern versions of Tcl. On the other hand, it does simplify the implementation a lot.

5. System Requirements

- Supports Windows 2000 [47] or higher, including the 64-bit varieties.
- Can be deployed using "xcopy" [41] or installed using the provided setup.
- Requires only the .NET Framework Version 2.0 [48].

These are the base requirements; however, the .NET Framework Version 2.0 Service Pack 1 [49] or higher is recommended. Also, having the .NET Framework 2.0 Software Development Kit [50] and/or Visual Studio 2005 [51] or higher available is very useful.

- Tcl/Tk integration requires a threaded build of Tcl 8.4 or higher.

Installed instances of ActiveTcl [52] are detected automatically.

6. Design Philosophy

Tcl heavily influenced the design of Eagle. In particular:

- Everything is a string.
- Commands are free to interpret their arguments however they wish.
- The language supplies primitives that can be combined in useful ways.

However, there are some differences in the design that reflect the differences in the underlying platforms:

- No per-thread data; all state is stored in the interpreter.
- No interpreter result.

Having no interpreter result merits special attention because it significantly reduces the coupling between components.

7. General Features

- Supports almost all of the core command set for Tcl 8.4 (approximately 90%).
- Supports Unicode [43] (UCS-2) [44] natively.
- Supports interactive debugging with single-step mode, breakpoints, variable watches, state examination/modification, isolated evaluation, and scripting support via the **`debug`** command.
- Supports script cancellation, runaway script detection, asynchronous events, and queuing of scripts.
- Supports read-only variables, commands, procedures, and interpreters.
- Supports both per-variable and interpreter-wide variable tracing.
- Supports the **`apply`** command from Tcl 8.5.
- Supports closures [45] via the new **`scope`** command.
- Supports integrating with Tcl/Tk via the **`tcl`** command, TIP 285 [42] ready, "stardll" compliant.
- Supports database access via the **`sql`** command.
- Supports automation of any object in the CLR via the **`object`** command.
- Performs automatic reference counting and cleanup of object references stored in variables.
- Transparent handling of event delegates.
- Supports calling any function in any dynamic link library (DLL) [38] via the **`library`** command.

8. Differences from Tcl/Tk

- No Tk functionality.
- No argument expansion syntax.
- No namespace support except the global namespace.
- No asynchronous input/output.
- No server sockets.
- No slave interpreters, no hidden commands, no aliases, and no Safe Tcl.
- The following Tcl commands are not implemented: **`binary`**, **`dde`**, **`fblocked`**, **`fcopy`**, **`fileevent`**, **`format`**, **`glob`**, **`history`**, **`memory`**, **`registry`**, **`scan`**, and **`trace`**.
- No Tcl library, such as the http, msgcat, and tcltest packages.
    - This means that the following Tcl library routines are not implemented: **`auto_execok`**, **`auto_import`**, **`auto_load`**, **`auto_mkindex`**, **`auto_mkindex_old`**, **`auto_qualify`**, **`auto_reset`**, **`pkg::create`**, **`pkg_mkIndex`**, **`tcl_endOfWord`**, **`tcl_findLibrary`**, **`tcl_startOfNextWord`**, **`tcl_startOfPreviousWord`**, **`tcl_wordBreakAfter`**, and **`tcl_wordBreakBefore`**.
- 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.
- 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.
- For the **clock format** command, some of the Tcl formatting characters are not supported.
- For the **clock scan** command, recognition of relative date/time strings (e.g. "1 day ago", "next Wednesday", etc.) is not supported.
- For the **fconfigure** command, all options except "**-encoding**" and "**-translation**" are not supported.
- For the **regexp** command:
    - The "**-about**" option is not supported.
    - 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.
- For the **pid** command, supplying the optional *channelId* argument is not supported (always returns empty string).
- For the **proc** command, arguments with default values are not supported.
- For the **exit** command, by default does not exit the process; it merely prevents further trips through the engine and the interactive loop.
- For the **after idle** command, we evaluate the script immediately prior to the next evaluation because we have no idle detection.
- For the **array** command:
    - The "**statistics**" sub-command is not supported.
    - The search sub-commands (i.e. "**anymore**", "**nextelement**", "**donesearch**", "**startsearch**") are not supported.
- For the "**env**" array, the **array names**, **array get**, and **info exists** commands are not supported.
- Documentation of the integration and extensibility API is incomplete.
- Support for **tcltest** functionality is incomplete.
- Unit tests are incomplete.

The following commands have no counterpart in the Tcl core command set:

- The **base64** command handles encoding and decoding Base64 [55] encoded data.
- The **debug** command allows scripts to communicate with the script debugger.
- The **do** command implements the "do while" control structure.
- The **guid** command handles creating, parsing, and validating Globally Unique Identifiers (GUID) [56].

- The **`hash`** command handles hashing data using cryptographic hash functions such as MD5 [57], SHA1, and SHA512 [58].
- The **`throw`** command allows scripts to throw exceptions.
- The **`uri`** command handles building and parsing of Uniform Resource Identifiers (URI) [59].
- The **`xml`** command handles serialization of objects to XML [62].
- The **`host`** command allows scripts to communicate with the host application via the IHost interface.
- The **`invoke`** command invokes a command or procedure and passes the specified arguments unmodified.
- The **`library`** command allows native library functions to be declared, called, and managed.
- The **`object`** command allows managed objects to be created, used, and disposed.
- The **`scope`** command allows call frames and their variables to be created and manipulated.
- The **`sql`** command allows access to ADO.NET [60] accessible databases.
- The **`try`** command implements the "try finally" control structure.
- The **`kill`** command allows processes to be terminated.

The following commands have additional options:

- The **`regsub`** command has an "**`-eval`**" option to evaluate a script for every match.
- The **`switch`** command has a "**`-substring`**" option to perform prefix matching (e.g. pattern "foo" matches string "foobar").
- The **`switch`** command has a "**`-subst`**" option to perform substitutions on a pattern prior to matching against it.
- The **`rename`** command has a "**`-nodelete`**" option to allow a command or procedure to be renamed to an empty string without being deleted.
- The **`exit`** command has a "**`-force`**" option to exit the process instead of merely invalidating the interpreter.

The **`file`** command has several new subcommands:

- The "**`information`**" subcommand to retrieve platform specific information about a file or directory.
- The "**`rights`**" subcommand to determine the available access rights for a file or directory.
- The "**`same`**" subcommand to determine if two file or directory names refer to the same physical file or directory.
- The "**`sddl`**" subcommand to return or set the security descriptor for a file or directory using the Security Descriptor Definition Language (SDDL) [61].

- The "**`version`**" subcommand to return the version information for an executable file.

The **`exec`** command has several new options:

- The "**`-directory`**" option to set the working directory for the process.
- The "**`-exitcode`**" option to specify the variable name where the process exit code should be stored.
- The "**`-nocapture`**" option to prevent capturing the process exit code and output.
- The "**`-nocarriagereturns`**" option to remove carriage returns from captured output.
- The "**`-shell`**" option to launch the process using the shell.
- The "**`-success`**" option to specify a successful exit code for the process other than zero.
- The "**`-stderr`**" option to specify the variable name where the output captured from the standard error channel should be stored.
- The "**`-stdout`**" option to specify the variable name where the output captured from the standard output channel should be stored.

9. Integration with the CLR, WinForms, WPF, and ASP.NET

Integration with any managed class is accomplished using the **`object`** command. Arguments and return values are automatically marshaled to and from their string representations. Opaque object handles represent return values and the values of arguments passed by reference when they are of types that are not readily convertible to string.

The marshaler understands the following data types, including their reference and nullable variations, where applicable:

Type	Handling
`System.Boolean`	Numeric or any "boolean string" (e.g. "true", "false", etc).
`System.SByte`	Numeric in the range -128 to 127.
`System.Byte`	Numeric in the range 0 to 255.
`System.Char`	Numeric in the range 0 to 65535.
`System.Int16`	Numeric in the range -32768 to 32767.
`System.UInt16`	Numeric in the range 0 to 32768.
`System.Int32`	Numeric in the range -2147483648 to 2147483647.
`System.UInt32`	Numeric in the range 0 to 4294967295.
`System.Int64`	Numeric in the range -9223372036854775808 to 9223372036854775807.
`System.UInt64`	Numeric in the range 0 to 18446744073709551615.

Type	Handling
`System.Decimal`	Numeric in the range $\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$.
`System.Single`	Numeric in the range $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$.
`System.Double`	Numeric in the range $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$.
`System.DateTime`	Dates and time strings from 12:00:00 AM, January 1, 0001 to 11:59:59 PM, December 31, 9999 (in 100-nanosecond units) or any numeric in the range 0 to 3155378975999999999.
`System.Enum`	The string name or numeric value of any defined value in the enumeration.
`System.Guid`	Any string that can be parsed by the CLR as a System.Guid.
`System.Uri`	Any string that can be parsed by the CLR as a System.Uri.
`System.Version`	Any string that can be parsed by the CLR as a System.Version in the format: "*major***.***minor*[**.***build*[**.***revision*]]".
`System.Delegate`	Any string that can be parsed as a script. Only delegates derived from or with a method signature that matches System.EventHandler are supported. Scripts can be registered and unregistered for events of an object using the "add_<eventName>" and "remove_<eventName>" methods of that object, respectively.
`System.Object`	Any string that represents an opaque object handle available in the interpreter. All types are supported because all other types derive from this type.
`System.Type`	Any string that represents the name of a type available in the interpreter. Names do not have to be fully qualified if the namespace has been imported via **`object import`**.
`System.Array`	Elements may be of any supported type, nested and multi-dimensional arrays are supported.
`StringList`	Any string that can be parsed as a Tcl-style list. Return values that are a single-dimensional array of a primitive type are converted to this type. This type is from the Eagle library.

10. Integration with the Host Application

The IHost interface is an integration point that allows the host application to customize how interactive input and output are handled. It also allows interception of requests for library scripts (e.g. “init.eagle”).

All methods of the IHost interface must be synchronous; therefore, it may be necessary to use a thread other than the primary user-interface thread to handle incoming IHost method calls.

11. Integration with Native Libraries

Integration with any native library function is accomplished using the **`library`** command. Arguments and return values are automatically marshaled to and from their string representations. Here is an explanatory example:

```
set GetStdHandle [library declare -functionname GetStdHandle \
        -returntype IntPtr -parametertypes [list UInt32] -module \
        [library load kernel32.dll]]

puts stdout [library call $GetStdHandle -11]; # STD_OUTPUT_HANDLE
```

12. Extensibility

The primary mechanism for adding functionality is through the creation of new commands. You do this by implementing the ICommand interface, and registering the implementation using the AddCommand method of the Interpreter object.

The IExecute interface is a subset of the ICommand interface. There is a default implementation of the ICommand interface, but for the command to actually do anything, it must implement the IExecute interface, which consists of a single method named “Execute” defined as:

```
ReturnCodes Execute(
        IInterpreter interpreter,
        IClientData clientData,
        ArgumentList arguments,
        ref Result result);
```

Commands and binary packages are allowed, though not required, to maintain internal state, and will be notified when they are being initialized or terminated through the IState interface.

If the command exists as part of a set of commands, a binary package can be created that implements the IPlugin interface to facilitate managing those commands as a single unit. The default implementation of the IPlugin interface automatically adds all commands implemented in that binary package to the interpreter.

Custom math functions are also supported. You can define functions by implementing the IFunction interface and registering it using the AddFunction method of the Interpreter object.

The IExecuteArgument interface is a subset of the IFunction interface. There is a default implementation of the IFunction interface, but for the function to actually do anything, it must implement the IExecuteArgument interface, which consists of a single method named “Execute” defined as:

```
ReturnCodes Execute(
        IInterpreter interpreter,
        IClientData clientData,
        ArgumentList arguments,
```

```
                ref Argument value,
                ref Result error);
```

Sample projects are provided with the source distribution demonstrating most of the available extensibility mechanisms.

13. Tcl/Tk Integration

Seamless integration with Tcl and Tk is made possible through the use of the **tcl** command. The following subcommands are defined:

Command	Description
tcl cancel ?-unwind? ?--? *path* ?*result*?	Initiates cancellation of the script being evaluated in interpreter identified by *path*.
tcl canceled *interp*	Returns an empty string if the script being evaluated in the interpreter identified by *interp* has not been canceled; otherwise, an error is generated.
tcl command create *srcCmd interp targetCmd*	Arranges for the Tcl command *targetCmd* to refer to the Eagle command *srcCmd* in the Tcl interpreter identified by *interp*.
tcl command delete *interp targetCmd*	Deletes the previously registered Eagle command association for the Tcl command *targetCmd* in the Tcl interpreter identified by *interp*. If no such association exists, an error is generated.
tcl command exists *interp targetCmd*	Returns non-zero if the specified Tcl command *targetCmd* refers to an association with an Eagle command in the Tcl interpreter specified by *interp*.
tcl convert *interp string type*	Attempts to convert *string* to the registered object type *type* in the interpreter specified by *interp*.

Command	Description
tcl create ?*options*?	Creates a Tcl interpreter and arranges for the Tcl command "**eagle**" to refer to the Eagle command "**eval**". If the "**-nobridge**" option is present, the "**eagle**" command is not created in the interpreter. If the "**-noinitialize**" option is present, library initialization will be skipped for the interpreter. If the "**-safe**" option is present, all known potentially-unsafe core functionality (both commands and variables) will be removed from the interpreter.
tcl delete *interp*	Deletes the interpreter identified by *interp*. If the interpreter is currently in use an error is generated.
tcl eval *interp arg* ?*arg* ...?	Concatenates the arguments, evaluates the result as a Tcl script in interpreter identified by *interp*, and returns the value.
tcl exists *interp*	Returns non-zero if the interpreter identified by *interp* exists.
tcl expr interp arg ?arg ...?	Concatenates the arguments and, evaluates the result as a Tcl expression in the interpreter identified by *interp*, and returns the value.
tcl find ?*pattern*?	Attempts to find all installed instances of the Tcl runtime library. If one or more instances are found they are returned as a list of fully qualified file names filtered by *pattern*; otherwise, an error is generated.

Command	Description
tcl load ?options? ?path?	Loads the Tcl runtime library. If *path* is not supplied, a search for all installed instances of the Tcl runtime library is performed and then the highest version available is loaded; otherwise *path* is treated as the path and file name of the Tcl runtime library to load. If the "**-flags**" option is present, it can be used to restrict the Tcl runtime library search. Please refer to the FindFlags enumeration for the complete list of legal values and their meanings. If the "**-minimumversion**" option is present, it is the minimum acceptable version of the Tcl runtime library. If the "**-maximumversion**" option is present, it is the maximum acceptable version of the Tcl runtime library. If the "**-unknownversion**" option is present, it is the version to use during the Tcl runtime library selection process when the version cannot be determined based on the file name.
tcl master	Returns the master interpreter. If the Tcl runtime library has not been loaded, an error is generated.
tcl ready ?*interp*?	Returns non-zero if the Tcl runtime library has been loaded and optionally checks if the interpreter identified by *interp* exists and is valid.
tcl resetcancel ?*options*? *interp*	Resets the script cancellation flags for the interpreter identified by *interp*. If the "**-force**" option is present, this is done even if the interpreter is currently in use.
tcl set *interp varName* ?*newValue*?	Returns or sets the value of the variable *varName* in the interpreter identified by *interp*.
tcl source *interp fileName*	Evaluates file *fileName* in the interpreter identified by *interp*.

Command	Description
tcl subst ?*options*? *interp string*	Performs backslash, command, and variable substitutions on *string* in the interpreter identified by *interp*. If the "**-nobackslashes**" option is present, backslash substitutions are not performed. If the "**-nocommands**" option is present, command substitutions are not performed. If the "**-novariables**" option is present, variable substitutions are not performed.
tcl types *interp*	Returns the list of registered types for the interpreter identified by *interp*.
tcl unload	Unloads the Tcl runtime library. If it is not loaded or if any interpreter is currently in use, an error is generated.
tcl unset *interp varName*	Unsets the variable *varName* in the interpreter identified by *interp*.
tcl update ?*options*?	Processes pending events. If the "**-wait**" option is present, block until an event is processed. If the "**-all**" option is present, all available events are processed before returning.

14. Limitations

Performance is the primary limitation at this point; some operations are up to two orders of magnitude slower than Tcl; however, there is certainly some room for performance gains through optimization.

Some commands provided by Tcl 8.4 are missing; however, plans are in place to implement most of the missing commands.

Namespaces are not supported. There are no namespaces except the global namespace. Any attempt to refer to a non-global namespace via the **namespace** command generates an error; however, using qualified variable names does not generate an error. The command "*set foo::bar 1*" actually creates a variable in the current call frame named "*foo::bar*".

15. Future Directions

In the near term:

- Implement asynchronous input/output and server sockets.
- Implement most of the missing commands from Tcl 8.4.
- Complete the documentation.

- Complete the unit testing framework and port the necessary tests from Tcl.

In the long term:

- Possibly implement the **`send`** command.
- Possibly support loading binary packages into isolated application domains [54].

In the deep and mysterious future:

- Possibly support full integration with Visual Studio.

16. Questions and Answers

How big is the project?

There are approximately 600 files in 120 directories. There are approximately 90 thousand physical lines of source code, 43 thousand logical lines of source code, or 35 thousand lines of IL code.

Where is the project hosted?

The project [34] is currently hosted on CodePlex [35]; however, there is currently no public repository due to an impedance mismatch between what they offer and what is actually required.

What is the license for the project?

Eagle is licensed [36] under the same terms as Tcl/Tk [37].

Can I help?

Certainly, anybody is welcome to contribute to the project.

What tasks remain prior to the final 1.0 release?

Fix any bugs encountered during the beta period, write documentation, and add more unit tests. No new features are planned before the final 1.0 release.

17. Conclusion

The initial scope of this project was modest; however, it eventually became clear that Eagle could be used as the hub of a universal integration platform for any managed application. It is being released as open-source after more than a year in development in the hopes that people will find it useful.

18. References

[1] Eagle, http://eagle.to/
[2] Tcl Developer Xchange, http://www.tcl.tk/
[3] Common Language Runtime (CLR), http://en.wikipedia.org/wiki/Common_Language_Runtime
[4] The C# Programming Language, http://msdn microsoft.com/vcsharp/
[5] Jacl, http://tcljava.sf net/
[6] Tcl/Tk 8.4, http://www.tcl.tk/software/tcltk/8.4 html
[7] Tcl/Tk 8.5, http://www.tcl.tk/software/tcltk/8.5 html
[8] Tcl/Tk 8.6, http://www.tcl.tk/software/tcltk/8.6 html
[9] Dynamic Language Runtime (DLR), http://en.wikipedia.org/wiki/Dynamic_Language_Runtime
[10] Open Source Initiative, http://www.opensource.org/
[11] The C Programming Language, http://en.wikipedia.org/wiki/C_(programming_language)
[12] The C++ Programming Language, http://en.wikipedia.org/wiki/C%2B%2B
[13] Java Virtual Machine (JVM), http://en.wikipedia.org/wiki/Java_Virtual_Machine
[14] The Java Programming Language, http://java.sun.com/

[15] Java Native Interface (JNI), http://java.sun.com/j2se/1.5.0/docs/guide/jni/
[16] Platform Invoke (P/Invoke),
[17] Visual Studio Tools for Applications (VSTA), http://msdn microsoft.com/vsx2008/products/
[18] Visual Basic for Applications (VBA), http://msdn.microsoft.com/isv/
[19] Visual Basic.NET, http://msdn.microsoft.com/vbasic/
[20] Visual Studio Tools for Applications FAQ, http://www.summsoft.com/content/faq.aspx
[21] Domain Specific Languages (DSL), http://msdn microsoft.com/vsx/
[22] Windows Presentation Foundation (WPF), http://windowsclient.net/
[23] IronPython, http://www.codeplex.com/IronPython
[24] Python, http://www.python.org/
[25] IronRuby, http://rubyforge.org/projects/ironruby
[26] Ruby, http://www.ruby.org/
[27] IronLisp, http://www.codeplex.com/IronLisp
[28] Lisp, http://www.lisp.org/
[29] IronScheme, http://www.codeplex.com/IronScheme
[30] Scheme, http://www.schemers.org/
[31] Nua, http://www.codeplex.com/Nua
[32] Lua, http://www.lua.org/
[33] Managed Jscript, http://en.wikipedia.org/wiki/Managed_JScript
[34] Eagle (CodePlex), http://www.codeplex.com/eagle
[35] CodePlex, http://www.codeplex.com/
[36] The Eagle License, http://eagle.to/standard/license html
[37] Tcl/Tk License Terms, http://www.tcl.tk/software/tcltk/license.html
[38] Dynamic link library, http://en.wikipedia.org/wiki/Dynamic-link_library
[39] Managed C++, http://msdn.microsoft.com/visualc/
[40] Global Assembly Cache, http://en.wikipedia.org/wiki/Global_Assembly_Cache
[41] XCOPY Deployment, http://en.wikipedia.org/wiki/XCOPY_deployment
[42] TIP 285, http://tip.tcl.tk/285
[43] Unicode, http://www.unicode.org/
[44] UCS-2, http://en.wikipedia.org/wiki/Universal_Character_Set
[45] Closures, http://en.wikipedia.org/wiki/Closure_(computer_science)
[46] Endekalogue, http://www.tcl.tk/man/tcl8.4/TclCmd/Tcl.htm
[47] Microsoft Windows 2000, http://www.microsoft.com/windows2000/
[48] Microsoft .NET Framework Version 2.0 RTM (x86),
http://www microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5
[49] Microsoft .NET Framework Version 2.0 Service Pack 1 (x86),
http://www microsoft.com/downloads/details.aspx?familyid=79bc3b77-e02c-4ad3-aacf-a7633f706ba5
[50] Microsoft .NET Framework 2.0 Software Development Kit (x86),
http://www microsoft.com/downloads/details.aspx?familyid=fe6f2099-b7b4-4f47-a244-c96d69c35dec
[51] Microsoft Visual Studio 2005, http://msdn microsoft.com/vs2005/
[52] ActiveTcl, http://www.activestate.com/Products/activetcl/
[53] startdll, http://wiki.tcl.tk/15969
[54] Application Domain, http://en.wikipedia.org/wiki/Application_Domain
[55] Base64, http://en.wikipedia.org/wiki/Base64
[56] Globally Unique Identifier (GUID), http://en.wikipedia.org/wiki/Globally_Unique_Identifier
[57] MD5, http://en.wikipedia.org/wiki/MD5
[58] SHA1, SHA512, http://en.wikipedia.org/wiki/SHA_hash_functions
[59] Uniform Resource Identifier (URI), http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[60] ADO.NET, http://msdn.microsoft.com/system.data.aspx
[61] Security Descriptor Definition Language, http://msdn.microsoft.com/library/aa379567(VS.85).aspx
[62] Extensible Markup Language, http://www.w3.org/XML/