APWTCL
(A Lite Weight Tcl Interpreter for cell phones)

History

- Start was about March 2012
- Based on itclinjavascript and Jim Tcl
- That version version was also written in javascript
- Looking for a version to run under Android
- Source code should be as similar as possible

Alternatives

- Use a javascript interpreter and run the javascript version
- Proved to be very slow
- In that version there are 3 levels of interpretation:

  - Java

  - javascript

  - Tcl
- Wanted to know how fast/similar a Java version would be

Design of Code

- Use Classes for:
  - TclObj
  - all the obj_types
  - all the core commands
- Build ScriptTokens when parsing script
- Evaluation mechanism for ScriptTokens
- Core commands are classes
- Core commands can have subcommands (ensembles)
- Conversions depending on wanted OBJ_TYPE

Example of objtypes

- OBJ_TYPE_ARRAY
- OBJ_TYPE_DICT
- OBJ_TYPE_INT
- OBJ_TYPE_ITCL
- OBJ_TYPE_NAMESPACE
- OBJ_TYPE_SCRIPT
- OBJ_TYPE_SOURCE
- OBJ_TYPE_STATEMENT
- OBJ_TYPE_STRING

The parser

- Parser based on dodekalogue
- Parser does tokenizing
- Tokens are stored in a TclObj
- TclObj has methods to convert to different objtypes
- Parser takes first token as command and executes with parameters
- Result code is as in Tcl (OK, ERROR, BREAK, RETURN, …)
- Result TclObj for returning the result of the execution

Callframes

- Very similar to C-Implementation of Tcl
- Container for all local variables
- Container for info about proc/function calls
- Reference to current namespace
- Reference Itcl object (if that is an Itcl call)
- Type of call (PROC,METHOD, UPLEVEL, EVAL, UPVAR, ...)

Namespaces (1)

- Very similar to C-Implementation of Tcl
- Provide namespace type for later implementation of Itcl
- Provide info about functions/variables for Itcl
- Provide mechanism for resolvers (command/variable)
- Reference for parent namespace
- Reference for child namespaces
- Namespace type:
  - NAMESPACE
  - ITCL_CLASS
  - ITCL_EXTENDEDCLASS
  - TYPE_CLASS

Namespaces (2)

- Lookup functionality:
  - Procs as in Tcl implementation
  - Methods as in Tcl/Itcl using resolver
  - Variables:
    - Reimplementation of Tcl's C-Implemenation:
      - LookupVariableEx
      - LookupSimpleVariable
      - GetNamespaceForQualName
      - FindNamespaceVar
    - Using resolver

Tcl command (proc)

- TclCommand object containing info
- Is built when parsing a proc statement
- Contains:
  - argument list
  - body
- When called:
  - build current arguments
  - be aware of special args argument
  - add default values for optional arguments if needed
  - store info in callframe stack
  - call proc (execute body) with arguments in callframe
  - return result

Itcl method implementation

- For Itcl object use ItclCommand similar to a TclCommand
- It's an adapter which does the following:
  - check in the namespace, if the subcommand exists
  - provide info on the callframe stack about the itcl object
  - switch to the relevant namespace
  - prepare the argument list as for TclCommand in callframe
  - call the method (execute body)
  - return the result

Tokenizing

- Idea:
  - technique „borrowed“ from Jim Tcl
  - do tokenizing without expanding (ParseToken)
  - Convert ParseTokens to ScriptTokens
  - use that info for later execution
  - expand necessary stuff based on ScriptTokens directly before executing
  - Example:
    - puts hello
    - set $i xy [foo]BAR

ScriptToken Example

- ScriptTokens:
  - TOKEN_LINE 2
  - TOKEN_ESC puts
  - TOKEN_ESC hello
  - TOKEN_LINE 4
  - TOKEN_ESC set
  - TOKEN_VAR i
  - TOKEN_WORD 2
  - TOKEN_VAR x
  - TOKEN_VAR y
  - TOKEN_WORD 2
  - TOKEN_CMD foo
  - TOKEN_ESC BAR

Evaluation of Statements

- Execute every command sequentially until end of script or error
- First token of the line is always TOKEN_LINE
- Populate the arguments objects
- Fast path if token does not need interpolation (word_tokens == 1)
- Interpret token
  - for TOKEN_COMMAND and TOKEN_BRACE call recursively
  - otherwise call substOneToken
- If word_tokens != 1 call interpolateTokens
- Get command from first token
- Interpret proc body if PROC, else call native implemented command
- Check for error and return relevant return code

Commands with conditionals

- Some commands need conditionals:
  - if
  - while
- Use of expression handler (same as for [expr] command)
- Makes these commands easy to implement

Android Version

- Written in Java
- Converted from Javascript version as similar as possible
- All Tcl core commands are an own class in java
- Dynamic loading and instantiating of core commands when needed
- Core commands are precompiled Java modules

iOS Version

- Written in Objective C
- Converted from Java version as similar as possible
- All Tcl core commands are an own class in Objective C
- Dynamic instantiating of core commands when needed
- Core command classes are loaded as Objective C classes

Status

- Time frame Javascript version Nov. 2011 - Feb. 2012 (30.000 LOC)
- Time frame Java version February to March 2012 (30.000 LOC)
- Time frame Objective C version April to May 2012 (45.000 LOC)
- Support of itcl classes (class, extendedclass, type, macro) and objects
- Support of [trace] command
- Support of [upvar] and [uplevel] command
- Support of namespaces
- Support of basic functionality of [tcltest]
- Support of [package] command (with pkgIndex.tcl)
- Good base for starting with Tk

Todos

- Test suite
- Complete integration of a lot of subcommands
- Additional commands
- Prepare alpha version
- Take care of feedback to that version
- Documentation
- Demos

Demo URL's

- Sorry none