Itcl in Javascript

An implementation of Tcl/Itcl using Javascript.

A paper for the Eighteenth Annual Tcl/Tk Conference

Abstract

Incr Tcl in Javascript (also called: itcl in Javascript) is a work in progress, which started about February 2011. It's intention is to extend the existing [Tcl in Javascript] an interpreter for the Tcl language written in Javascript with a lot of additional features and commands as well as an implementation of itcl in Javascript. During implementation there was the need for optimizing parsing and evaluation of Tcl statements, which resulted in a partial parsing strategy.

Contact information

Arnulf Wiedemann

Lechstr. 10

D-86931 Prittriching

Email: arnulf@wiedemann-pri.de

Inhaltsverzeichnis

Abstract....2
Contact information....3
1The Idea....6
2How it started....7
3Design Goals for Implementation of Itcl in Javascript....9
4Performance issues....16
5Tcl Javascript Objects....17
5.1 [TclCallframe js Object]....17
Parameters:....17
5.2 [TclCommand js Object]....17
Parameters:....17
5.3 [TclDict js Object]....18
Parameters:....18
5.4 [TclEvalStatement js Object]....18
Parameters:....18
5.5 [TclInterpAlias js Object] 18
Parameters:....18
5.6 [TclInterp js Object]....18
Parameters:....18
5.7 [TclNamespace js Object]....19
Parameters:....19
5.8 [TclNode js Object]....19
Parameters:....19
5.9 [TclObject js Object]....19
Parameters:....19
5.10 [TclPackage js Obejct]....20
Parameters:....20
5.11 [TclParser js Object]....20
Parameters:....20
5.12 [TclParseStatement js Object] 20
Parameters:....20
5.13 [TclResolve js Object]....20
Parameters:....20
5.14 [TclStatement js Obejct]....20
Parameters:....20
5.15 [TclTest js Object]....21
Parameters:....21
5.16 [TclTestResult js Object]....21
Parameters:....21
5.17 [TclTrace js Object]....21
Parameters:....21
5.18 [TclVariable js Object]....21
Parameters:....21
5.19 [TclWord js Object]....22
Parameters:....22
5.20 [TclWordPart js Object]....22
Parameters:....22
6Itcl Javascript Objects for Tcl....23
6.1 [ItclClasses js Object]....23

Parameters: 23
6.2 [ItclClass js Object] 23
Parameters: 23
6.3 [ItclCommand js Object] 23
Parameters: 23
6.4 [ItclFunction js Object] 23
Parameters: 23
6.5 [ItclFunctionParam js Object] 24
Parameters: 24
6.6 [ItclObject js Object] 24
Parameters: 24
6.7 [ItclOption js Object] 24
Parameters: 24
6.8 [ItclVariable js Object] 24
Parameters: 24
6.9 TclItclDict 25
Parameters: 25
6.10 TclItclHelper 25
Parameters: 25
7Status 26

1The Idea

During looking for a suitable tool to run Tcl in a browser I found [Tcl in Javascript] also named JsTcl. I found it interesting but with too few features from Tcl, so I decided to enhance that. JsTcl is an implementation of a Tcl interpreter written in Javascript and the parser is based on [Picol]. An implementation in Javascript has the big advantage, that it is running on most browsers, also there are some incompatibilities to take care of between the browsers. One advantage is, that you don't need to compile and link and besides the browser incompatibilities you don't have to worry about the platform you are running on.

2How it started

When looking for a frontend/client for [ATWF] and [Reporting Tools with Tcl] I did spend some time checking all the available stuff. For a [Tcl] client it was easy there I decided to use a [Tclkit]/[Starkit], but in a browser, the only existing possibility is the [Tcl/Tk plugin], but there I knew that there were some problems and it was not much updated in the past.

So I was thinking about generating [HTML] code on the server side with [Tcl]. At that time I was also checking what of the existing [javascript] based libraries could be used especially [jQuery] and [YUI]. Both had a lot of interesting features. I also found [Tcl in Javascript] and was thinking about using that as an interface for making either [jQuery] or [YUI] functionality available with a [Tcl] wrapper.

When looking closer at that and playing with it I did like it a lot, but was soon missing [Tcl] functionality I wanted to use.

That was the start of the [incr Tcl in Javascript] project. At the beginning I was only adding some (from my point of view) functionality, which I was missing most.

The general idea already used/implemented by Stéphane Arnold was to have TclObjects (TclObj) implemented as [javascript Object]s ([TclObject js Object]), which hold a [Tcl] value that can be converted into the different types needed like:

- string
- integer
- real
- boolean
- list

I have added to these types:

  - dict
  - stmt
  - word
  - word_part

The „dict“ type holds - as in the Tcl C-implementation - the internal representation of a Tcl dictionary.

I started with implementing [namespace]s. This was done by having a [javascript Object] [TclNamespace js Object] that did have properties for the relevant information and was used in a similar way as the TclNamespace struct in the C implementation. It contains references to the parent [namespace] and a list of the child namespace references. There was some basic implementation of a callframe available, I modified that to use a [TclCallframe js Object], which had additional properties like the currently executed statement for introspection, the used [namespace] for that callframe etc.

For [namespace]s also the [Tcl] parser ([TclParser js Object]) had to be extended to understand the [namespace] syntax for command and variable names.

The [TclNamespace js Object] was designed to allow different types of namespace:

•Tcl.NAMESPACE a “normal” [Tcl] [namespace]

•Tcl.ITCL_CLASS an [itcl] class [namespace]

•Tcl.ITCL_EXTENDED_CLASS an [itcl] extendedclass [namespace]

•Tcl.TYPE_CLASS an [itcl] type [namespace]

For the [itcl] namespace types there was designed a resolve_commands reference for allowing implementation of namespace command resolvers.

This Object includes method for registering class commands and for registering subcommands, which is used for implementing namespace ensembles. A list of superclasses is provided here as well as entries for a class constructor and a class destructor.

Later on there was added support for namespace variables which are handled with a reimplementation in [javascript] of the equivalent C-functions lookupVariableEx, lookupSimpleVariable, GetNamespaceForQualName and FindNamespaceVar.

3Design Goals for Implementation of Itcl in Javascript

The internal types of a [TclObject js Object] (in the C-Implementation a TclObj) are:

•OBJECT_TYPE_TEXT

•OBJECT_TYPE_LIST

•OBJECT_TYPE_INTEGER

•OBJECT_TYPE_REAL

•OBJECT_TYPE_BOOL

•OBJECT_TYPE_DICT

•OBJECT_TYPE_STMTS

•OBJECT_TYPE_STMT

•OBJECT_TYPE_WORD

•OBJECT_TYPE_WORD_PART

•OBJECT_TYPE_EXPR_TREE

Parsing rules for Tcl script input are corresponding to the [Dodekalog].

The reason for partially parsing the Tcl input is mostly performance and to some extent later on easier handling of the execution of a statement. Partially parsing is done in the following way: all the tokenizing for Tcl is done, but no variables are expanded, no bracket commands are executed and braced parts are handled as one token. And also within quoted strings the parts, which have later on to be expanded are parsed into separate “word_part” [TclObject js Object]s. Same is done for array names and array references.

Tokens returned from parsing are:

•TOKEN_WORD_SEP

•TOKEN_STR

•TOKEN_EOL

•TOKEN_EOF

•TOKEN_ESC

•TOKEN_CMD

•TOKEN_VAR

•TOKEN_EXPAND

•TOKEN_PAREN

- TOKEN_BRACE
- TOKEN_VAR_ARRAY
- TOKEN_VAR_ARRAY_NAME
- TOKEN_ARRAY_NAME
- TOKEN_VAR_COMPOSED
- TOKEN_BRACED_VAR
- TOKEN_QUOTED_STR
- TOKEN_COMMENT
- TOKEN_DECIMAL
- TOKEN_INTEGER
- TOKEN_REAL
- TOKEN_BOOLEAN
- TOKEN_HEX
- TOKEN_OCTAL
- TOKEN_MINUS
- TOKEN_PLUS
- TOKEN_MUL
- TOKEN_DIV
- TOKEN_MOD
- TOKEN_LT
- TOKEN_GT
- TOKEN_LE
- TOKEN_GE
- TOKEN_NE
- TOKEN_EQ
- TOKEN_NOT
- TOKEN_RP
- TOKEN_AND
- TOKEN_OR
- TOKEN_EXOR
- TOKEN_AND_IF
- TOKEN_OR_IF
- TOKEN_STR_EQ
- TOKEN_STR_NE
- TOKEN_STR_IN
- TOKEN_STR_NI

- TOKEN_STR_PARAM
- TOKEN_STR_CMD
- TOKEN_NO_WORD_SEP
- TOKEN_EXPR
- TOKEN_STMTS

For “normal” Tcl code the tokens from TOKEN_WORD_SEP to TOKEN_COMMENT are returned

Tokens TOKEN_DECIMAL to TOKEN_STR_NI are returned for expression like parts in if, while and in the expr command. The last few ones are used internally for partially parsed statements.

Examples:

•String	Token	Value
•$abc	TOKEN_VAR	abc
•${abc def}	TOKEN_BRACED_VAR	
	■TOKEN_BRACE	
	■TOKEN_STR	abc
•${abc def}	TOKEN_BRACED_VAR	
•x(y)	TOKEN_ARRAY_NAME	x0x01y
•$x(y)	TOKEN_VAR_ARRAY	
•${x}(y)	TOKEN_VAR_ARRAY_NAME	
•[set a 1]	TOKEN_CMD	
•{a y}	TOKEN_BRACE	a y
•“abc”	TOKEN_QUOTED_STRING	abc
•“abc[x a]$y{d e f}yyy”	TOKEN_QUOTED_STRING	
	■TOKEN_STR	abc
	■TOKEN_CMD	x a
	■TOKEN_VAR	y
	■TOKEN_BRACE	d e f
	■TOKEN_STR	yyy
•xyz	TOKEN_STR	xyz
•{*}	TOKEN_EXPAND	“”

Some commands use a statement part as en expression to be evaluated and to return a value of true or false like if and while for the condition or the Tcl expr command. For these the condition is parsed to an expression tree existing of nodes ([TclNode js Object]). When tokenizing an expression string first all parts are put into nodes objects and these [TclNode js Object]s are placed in a tree with the operator as the parent node and the operands as the child nodes. A paren “(“ is also a parent node. The nodes are first put in parsing order in the expression tree and afterward the expression tree is reorganized according to the precedence rules o the operators.

Operators are:

•+	TOKEN_PLUS	
•-	TOKEN_MINUS	
•*	TOKEN_MUL	
•/	TOKEN_DIV	
•%	TOKEN_MOD	
•<	TOKEN_LT	
•>	TOKEN_GT	
•<=	TOKEN_LE	
•>=	TOKEN_GE	
•!=	TOKEN_NE	
•==	TOKEN_EQ	
•!	TOKEN_NOT	
•(	TOKEN_PAREN	pseudo operator used for precedence handling
•)	TOKEN_RP	pseudo operator used for precedence handling
•&	TOKEN_AND	
•|	TOKEN_OR	
•^	TOKEN_EXOR	
•&&	TOKEN_AND_IF	
•||	TOKEN_OR_IF	
•eq	TOKEN_STR_EQ	
•ne	TOKEN_STR_NE	
•in	TOKEN_STR_IN	
•ni	TOKEN_STR_NI	

Precedence rules are (as in C):

TOKEN_OR_IF	1
TOKEN_AND_IF	2
TOKEN_OR	3
TOKEN_EXOR	4
TOKEN_AND	5
TOKEN_EQ	6
TOKEN_NE	6
TOKEN_LT	7
TOKEN_GT	7
TOKEN_LE	7
TOKEN_GE	7
TOKEN_PLUS	9
TOKEN_MINUS	9
TOKEN_MUL	10
TOKEN_DIV	10
TOKEN_MOD	10
TOKEN_PAREN	12
TOKEN_STR	99

Reorganizing is done in flipping nodes that have a higher precedence:

if precedence of node is greater than the precedence of the left node and the node is not a TOKEN_PAREN flip nodes.

- set the parent->child_left to child_left of the node
- set parent of the node to child_left
- set child_left of the node to child_left->child_right
- set child_left->child_right to the node
- reorganize child_left

Some tokens are used only internal during parsing:

TOKEN_EOL	the separator for a Tcl statement either “\r”, “\n” or a “;”
TOKEN_WORD_SEP	space or tab between words
TOKEN_ESC	for signaling the different parts of a word like in yyy[a b]ccc one part is yyy one part is the TOKEN_CMD “a b” and one part is ccc. In between TOKEN_ESC is returned to signal these parts
TOKEN_EOF	at the end of the code to parse

Some tokens are only used within expression trees:

TOKEN_INTEGER	1 to n digits 0-9
TOKEN_DECIMAL	an integer with a leading unary “+” or “-”
TOKEN_REAL	a decimal with a “.” as the fraction separator, a fraction and an exponent with e+/-nnn syntax
TOKEN_BOOLEAN	the Tcl values for a boolean, true/false/0/1 …
TOKEN_HEX	0x followed by 0-9A-Fa-f characters
TOKEN_OCTAL	0-7 1 to n characters

4Performance issues

The original implementation of Tcl in Javascript was parsing every statement byte by byte when executing Tcl code.

When starting I used that for a while too, but when trying to implement itcl in Tcl, there were big performance problems. One point was the complete parsing of a statement when executing Tcl code, second problem was having itcl implemented in Tcl, which forced a twice time interpretation when interpreting a class definition once by javascript for running Tcl second by Tcl for parsing itcl.

So I decided to write the itcl interpreter in javascript too and as a second issue for better performance to parse initial Tcl code only once and then keep something like an intermediate code a “partially parsed” form of Tcl statements.

Because of Tcl's dynamic structure, that partially parsed form did not extend variable references or commands in braces etc. but did mostly a tokenizing, so the low level parsing had to be done only once. That did help a lot to increase performance. Nevertheless it would be useful in the future to have something similar to the Tcl bytecode and to have an interpreter for that intermediate format written in javascript. Designing and implementing such an intermediate language and an interpreter for it could perhaps be a future [GSoC] project.

There are three types that are used for a partially parsed Tcl statement. A Tcl statement is mapped to a “stmt” type, the parts of a Tcl statement like the command name and the params are represented as „word“ (statement part) and because a statement part can be composed of different sub parts. A sub part - named a “word_part” - is available, that can be:

•a variable reference within a quoted string

•a name constructed form a string part for example an array name

•a command part in brackets etc..

To hold that parsed information there are 3 different [javascript Object]s:

•[TclStatement js Object]

•[TclWord js Object]

•[TclWordPart js Object]

see below for details.

Later on there was another improvement of performance in implementing a cache for variable and command access depending on the callframe and using some epoch mechanism like it is used in TclOO for determining if the cache is still up-to-date. As every command and variable object (javascript Object) has an id in it the maximum id for variables/commands is used to determine, if the cache is still usable, as creating a variable/command modifies that maximum id and renaming/deleting a variable/command has to remove the corresponding cache entries.

Implementing that cache mechanism together with more often using the partially parsed statements gave a performance improvement of 400% for simple setting or getting a variable! “There is still some room for improvements” - as one of my former colleagues used to say - concerning performance.

5Tcl Javascript Objects

5.1 [TclCallframe js Object]

Parameters:

- interp
- type

Container for a Tcl callframe. A callframe is a [javascript Object] which is pushed on a stack built from a [javascript Array] when a proc or method is called. Contains all local variables, these are in a [javascript Object]: variables as [TclVariable js Object]s and the name as the index, and information about a possible [itcl] object, when the command was an [itcl] object. There is also a type of a callframe, which can be – as in the C implementation -

- CALL_TYPE_PROC
- CALL_TYPE_METHOD
- CALL_TYPE_UPLEVEL
- CALL_TYPE_UPVAR
- CALL_TYPE_EVAL

After the call the callframe object is popped from the stack again.

5.2 [TclCommand js Object]

Parameters:

- name
- func
- privdata

A TclCommand represents either a Tcl proc or a Tcl command implemented as a javascript function. The func argument is either javascript implementation of a Tcl command or a Tcl sub command or it is the javascript implementation of the Tcl proc command and in that case the privdata argument contains an array with the arglist and the body. This object also contains a call function, which checks, if there are execution traces and adds the enterstep and leavestep traces, so the interp knows which ones to call and it evaluates the enter trace, if one exists.

Now the functions code is executed with the calling arglist. For javascript functions it just executes them, for a Tcl proc the relevant javascript implementation function is called. This pushes a new callframe onto the callframe stack and prepares the arglist for Tcl including handling of the special args argument. During that part also the contents of the arglist, which can be a braced word is extracted. The the handling of optional arguments is done with filling in default values, if the argument is not there. The arglist is the stored as local variables in the callframe, so one can set and get these variables using the formal parameter names.

Also the level info for the info level command is prepared and pushed onto the callframe. After that evaluation of the body is done.

Now the level_info is popped off the callframe and the callframe is popped off the callframe stack. Following this the leave and leavestep traces are taken of the interpreter and if there is a leave trace that is called with the result of the executed code.

5.3 [TclDict js Object]

Parameters:

- interp

A container for the functions which build the dict ensemble. The dicts itself are [TclObject js Object]s. The implementation is very similar to the C-implementation in that it stores the keys and values for the keys in an associative array and the sequence of the keys in and additional array. Every value can be another dict. Conversion between the dict representation and the string representation and vice versa is done in the [TclObject js Object] when needed.

5.4 [TclEvalStatement js Object]

Parameters:

- Interp
- statement_parser

Javascript functions to evaluate preparsed Tcl statements and words, if the latter for example is a braced command.

5.5 [TclInterpAlias js Object]

Parameters:

- src_path
- src_cmd
- target_path
- target_cmd
- params

A container for holding information on mapping a Tcl command to another Tcl command. Right now the src_path and target_path have to be the same (interpreter) and must be empty for the current interpreter. When looking up a command to call the interpreters alias list is first searched for a relevant command and after that the command is looked up in the namespace etc.

5.6 [TclInterp js Object]

Parameters:

- win
- start_dir

A container which holds all the information necessary for an Tcl interpreter like the stack for the callframes, the stack for the namspaces etc.

5.7 [TclNamespace js Object]

Parameters:

- interp
- ns_name
- privdata

A container for managing all the information for a Tcl namespace including an itcl class. With all the functions for looking up commands and variables and creating and deleting namespaces.

5.8 [TclNode js Object]

Parameters:

- interp
- name
- node_type
- child_left
- child_right

A container for holding a node of an expr tree used in if, while and expr Tcl command

5.9 [TclObject js Object]

Parameters:

- interp
- value
- type

A TclObject holds - as in the C implementation – information about a Tcl values. That can be a string, an integer, a dict and additionally here a statement, a word, an expr tree etc.

The type determines the initial type of the object. This type can change when shimmering.

Possible types are:

- OBJECT_TYPE_TEXT
- OBJECT_TYPE_LIST
- OBJECT_TYPE_INTEGER
- OBJECT_TYPE_REAL
- OBJECT_TYPE_BOOL
- OBJECT_TYPE_DICT
- OBJECT_TYPE_STMTS
- OBJECT_TYPE_STMT
- OBJECT_TYPE_WORD
- OBJECT_TYPE_WORD_PART
- OBJECT_TYPE_EXPR_TREE

5.10 [TclPackage js Obejct]

Parameters:

- interp

Container for holding all information of a Tcl or Tk package like the script for loading the package, the version umber etc. Including functions for the sub commands.

5.11 [TclParser js Object]

Parameters:

- Text

The base container for parsing Tcl statements. Also includes all the functions necessary to parse Tcl statements and get back the words and word parts of a Tcl statement..

5.12 [TclParseStatement js Object]

Parameters:

- interp

Container for parsing a normal Tcl statement or an expr of an if, while or expr Tcl command. It builds as a side effect an expr tree or the statements/statement/word/word_part info when parsing. Both cases use TclParser object to get the input parsed into tokens.

5.13 [TclResolve js Object]

Parameters:

- interp

- type

A container for resolving variable and function references for itcl classes/objects

5.14 [TclStatement js Obejct]

Parameters:

- interp
- file_name
- line_no
- word_obj

Container for holding info for a parsed Tcl statement. It contains a list of words, which in turn can contain a list of word_parts and both can contain statements (a list of statement info) when the statement contained a word which for example was a proc body.

5.15 [TclTest js Object]

Parameters:

- Interp

A container for the functions for building a tcltest test case and running it.

5.16 [TclTestResult js Object]

Parameters:

- interp
- test_name
- test_description
- expected_result
- result

A container for storing the result of a tcltest tcl test case, including an error message for a failing test etc.

5.17 [TclTrace js Object]

Parameters:

- interp
- type
- name

- ops
- command

A container for holding information for Tcl traces for variable, command and execution traces.

5.18 [TclVariable js Object]

Parameters:

- interp
- name
- type

A container for a Tcl variable contains a [TclObject js Object] and additional information like the type of the variable etc.

5.19 [TclWord js Object]

Parameters:

- interp
- token
- value
- file_name
- last_line_no
- line_no
- stmts

A container for a part of a Tcl statement. Can contain a list of statements, if it is for example the body of a proc.

5.20 [TclWordPart js Object]

Parameters:

- interp
- token
- value
- stmts

A container for information of parts of a TclWord, for example the parts of a string, when the string contains a variable reference or a braced command etc.

6Itcl Javascript Objects for Tcl

Objects used for implementing itcl classes and itcl objects. They hold the information needed to describe the itcl class/object

6.1 [ItclClasses js Object]

Parameters:

- interp

Contains references to all itcl classes/class objects (ItclClass).

6.2 [ItclClass js Object]

Parameters:

- interp
- name
- full_name
- class_type

Contains information about itcl class methods, variables, types, options etc.

6.3 [ItclCommand js Object]

Parameters:

- name
- class_name
- func_type
- protection
- func
- params
- body

This is the container an itcl method, same as TclCommand is for Tcl procs

6.4 [ItclFunction js Object]

Parameters:

- interp

6.5 [ItclFunctionParam js Object]

Parameters:

- interp
- definition
- min_args
- max_args
- have_args_arg
- usage
- default_args

Definition of the parameter signature of an itcl method/proc

6.6 [ItclObject js Object]

Parameters:

- Interp
- name
- class_obj
- constructor_args

Container for an itcl object, contains set of class variables for that object of all classes n inherited classes.

6.7 [ItclOption js Object]

Parameters:

- interp

A container for an itclextended class option (like a Tk option) with all the information about the name and class of the option, a possible default value, a possible script for cget, configure and validate or a possible variable containing a cget, configure or a validate method

6.8 [ItclVariable js Object]

Parameters:

- interp

Container for an itcl variable with the possible protection, init value and config script and the type (variable or common)

6.9 TclItclDict

Parameters:

- interp

still a leftover from the time when itcl basics were in Tcl and parsed from Tcl. Nowadays the itcl parsing is implemented in js too (deprecated should possibly be removed, have to check).

6.10 TclItclHelper

Parameters:

- Interp

still a leftover from the time when itcl basics were in Tcl and parsed from Tcl. Nowadays the itcl parsing is implemented in js too (deprecated should possibly be removed, have to check).

7Status

The interpreter is in the middle of the implementation, there are still a lot of sub commands missing and there is the need for test cases, as I know already about some problems/bugs and there will be a lot of bugs still in there, which have not yet been found because of missing test cases. Right now there has been spent more time to the second topic [Tk Widgets in Javascript] to be able in the near future to have some demos, which show some of the functionality. There is also the need for more examples/demos.