TCL Abstract Lists

What Are They, and
Why Extension Authors Should Care

Brian Griffin - July 20, 2023
Abstract Lists in TCL

Agenda

• Introduction
• What is a TCL List
• What is an Abstract List
• Why Abstract Lists
• How to Create an Abstract List
• Critical Things to Know! (even if you are not interested in Abstract Lists!)
• Q & A
Introduction

A quote from a recent post in TclChat:

mjanssen:

"I’ve worked with two tools that use TCL and implement the concept of "collections", which are basically sets of elements where each element has a very complex description. In one of them, collections had string representations so you could use them as lists, but transforming them into lists had a high cost. So my understanding is that the tool created versions of several commands (concat, lappend, lindex, etc) that bypassed TCL’s native implementation."
TCL List

• A collection of values stored in an C array.
• Access to values by indexing the array.
• Values stored in a Tcl_Obj struct
  • Tcl_Obj holds 2 representations, (called the ”stork” model)
    • Canonical string
    • Internal representation, such as an integer or a List.
• Access via commands in a script, or C API calls.
• List commands: lindex, lsearch, lsort, lrange, foreach, lset, etc, are tied to this data model.

0: ["One"]
1: ["Two"]
2: ["Three"]
3: ["Four"]
Abstract List

• An Abstract List
  • Separates the data management from the access operations.
  • Access is via a set of protocol functions.
• How list values are stored or managed depends on the protocol functions.
  • Examples: RB-Tree, Hash Table, Directed Graph, \( f(x) \)
• Values can even be computed on demand
  • The value for a given index must be consistent with the string representation of an equivalent List.
Abstract List
continued

• The \texttt{lseq} command is implemented as an Abstract List.

• Generates a sequence of numbers based on a start, end, and step values.

\begin{verbatim}
\texttt{lseq 10 .. 15 by 3} -> \{10 13 16 19 22 25\}
\end{verbatim}

• Values computed using math: \( f(index) = (\text{start} + (index \times \text{step})) \)

• This allows for very large lists with O(1) create time.
\textit{Just don’t ask for a string of the entire list. 😊}
WARNING!

Extension Maintainers — Take Note!

• A TCL List value holds a reference to an element Obj.
• Tcl_ListObjIndex(interp, listPtr, index, &elemObj) Returns an Obj with a refCount of 1 or greater.
• With an Abstract List, it is possible that the returned element can have a refCount of 0.
• lseq will return an element with a refCount of 0.
• The caller is responsible for freeing the Obj when it is no longer needed!
• Tcl_BumpObj(objPtr) - new function used to free Obj with refCount==0
• Or use the practice of calling Tcl_IncrRefCount(), then Tcl_DecrRefCount()
Abstract List
continued

• In some cases, an Abstract List will be converted to a traditional list.

• This occurs when the Abstract List is incapable of honoring the protocol request.

• An example: using \[lset\] on an \[lseq\] sequence:

  \[
  \text{set } s \ [lseq \ 8] \\
  \text{lset } s \ 5 \ "\text{Hi}\"
  \]

• After the \text{lset}, the new value will be a traditional list:

  \[
  0 \ 1 \ 2 \ 3 \ 4 \ \text{Hi} \ 6 \ 7
  \]
Why Abstract Lists

• Optimize
  • Value storage space
  • Value access
  • Computation
• Eliminate the need to mimic List commands
• Reduce or eliminate ”shimmering”
  • In TCL, defined scalar value types have a Length function that always returns 1
• This avoids the conversion to a List

```
set k [expr {6 + 7}]
set point {3 15}
⋯
if {llength $k] > 1} {
  # Process a point
  ⋯
} else {
  set x [expr {$k / 2}]
}
```
Why Abstract Lists
continued

• Example extensions that can benefit from Abstract Lists
  • tarray - https://wiki.tcl-lang.org/page/TArray
  • vectcl - http://auriocus.github.io/VecTcl/
    and http://auriocus.github.io/VecTcl/design/40.html
  • There are certainly many more …
Creating an Abstract List

List Protocol Functions

\[ \text{Tcl\_Size} \ (\text{LengthProc}) \quad (\text{Tcl\_Obj} *\text{listPtr}); \]
\[ \text{int} \quad \text{(IndexProc)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{Tcl\_Obj} *\text{listPtr}, \text{Tcl\_Size} \text{index}, \text{Tcl\_Obj}** \text{elemObj}); \]
\[ \text{int} \quad \text{(SliceProc)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{Tcl\_Obj} *\text{listPtr}, \text{Tcl\_Size} \text{fromIdx}, \text{Tcl\_Size} \text{toIdx}, \text{Tcl\_Obj} **\text{newObjPtr}); \]
\[ \text{int} \quad \text{(ReverseProc)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{struct} \text{Tcl\_Obj} *\text{listPtr}, \text{Tcl\_Obj} **\text{newObjPtr}); \]
\[ \text{int} \quad \text{(GetElements)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{struct} \text{Tcl\_Obj} *\text{listPtr}, \text{Tcl\_Size} *\text{objcptr}, \text{Tcl\_Obj} ***\text{objvptr}); \]
\[ \text{Tcl\_Obj}* \quad \text{(SetElement)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{Tcl\_Obj} *\text{listPtr}, \text{Tcl\_Size} \text{indexCount}, \text{Tcl\_Obj} *\text{const} \text{indexArray}[], \text{Tcl\_Obj} *\text{valueObj}); \]
\[ \text{int} \quad \text{(ReplaceProc)} \quad (\text{Tcl\_Interp} *\text{interp}, \text{Tcl\_Obj} *\text{listObj}, \text{Tcl\_Size} \text{first}, \text{Tcl\_Size} \text{numToDelete}, \text{Tcl\_Size} \text{numToInsert}, \text{Tcl\_Obj} *\text{const} \text{insertObjs}[]); \]
Creating an Abstract List

typedef struct Tcl_ObjType {
    const char *name;
    Tcl_FreeInternalRepProc *freeIntRepProc;
    Tcl_DupInternalRepProc *dupIntRepProc;
    Tcl_UpdateStringProc *updateStringProc;
    Tcl_SetFromAnyProc *setFromAnyProc;
    size_t version;
} Tcl_ObjType;

/* List emulation functions - ObjType Version 2 */
Tcl_ObjTypeLengthProc *lengthProc;  /* Return the [llength] */
Tcl_ObjTypeIndexProc *indexProc;  /* Return [lindex $al $index] */
Tcl_ObjTypeSliceProc *sliceProc;  /* Return [lrange $al $start $end] */
Tcl_ObjTypeReverseProc *reverseProc;  /* Return [lreverse $al] */
Tcl_ObjTypeGetElements *getElementsProc;  /* Return an objv[] of all elements */
Tcl_ObjTypeSetElement *setElementProc;  /* Replace element, as in [lset al $val] */
Tcl_ObjTypeReplaceProc *replaceProc;  /* Replace subset with subset, e.g. [lreplace] */
} Tcl_ObjType;
Creating an Abstract List

continued

• Set version field using a macro

  • TCL_OBJCTYPE_V0  // Used for existing definitions (pre 9.0)
  
  • TCL_OBJCTYPE_V1(a)  // Used for scalar values.
    // ”a” is an length function that returns 1
  
  • TCL_OBJCTYPE_V2(a,b,c,d,e,f,g)  // Used for Abstract Lists
    // a-g are the set of List functions
Creating an Abstract List

continued

- Abstract List protocol functions are counterparts to the C API functions that perform the same action.

- List protocol functions are optional*.

- In the absence of a function, TCL will revert back to internal List behavior.
  
  - For example, without a SetElement function, TCL will first convert the list to a traditional TCL list, then complete the [lset] operation as usual.

- *The Length function is required for any Abstract List. Without it, none of the other functions will be invoked.
Creating an Abstract List

Examples

• Example "toy" implementations of Abstract Lists can be found:
  • https://github.com/bgriffinfortytwo/abstractlist-toys

• An Abstract List adaptation of VecTcl
  • https://github.com/bgriffinfortytwo/VecTcl9/tree/abstractlist
WARNING!

- A TCL List value always hold a reference to an element Obj.

- There is code out there that relies on calls to Tcl_ListObjIndex(…) to return an Obj with a refCount of 1 or greater.

- The caller then does not bother with refCount management if it does not hold a reference directly. — This can result in a memory leak in TCL9.0!

- [lseq] will return an element with a refCount of 0. The caller is responsible for freeing the Obj when it is no longer needed!

- Tcl_BumpObj(objPtr) - new function used to free Obj with refCount==0
The Tcl_ObjType struct has new fields.

Code that uses Tcl_ObjType will likely receive a compile warning or error.

Use the macro TCL_OBJTYPE_V0 to correct the warning. No other changes will be needed.
Precursors of Abstract Lists

- TIP 225: Arithmetic Series with Optimized Space Complexity
- TIP 629: Add a lseq (formally "range") command

- The original authors of the Abstract Lists concept (and some code) are: Alexandre Ferrieux, Salvatore Sanfilippo, and Miguel Sofer