TclBin extension
Binary data manipulating support for **Tcl**.

Software & current documentation by Laurent Demailly
email: ld@mail.box.eu.org
Initial documentation by Dimitry Kloper
email: dimka@tochna.technion.ac.il

January 30, 1997

This documentation corresponds to the release 1.2 (eigth public release / January 1997) of the package (see `http:www.box.eu.org/~dl/tclbin.html`).

For reference, please see the TclBin(n) manual page.

Abstract

Generally, **Tcl** has only one data type – strings. This "everything is a string" conception is a matter of many debates relating **Tcl**'s speed and other subjects. A noticeable disadvantage of such mechanism is that no binary data can be processed effectively.

This extensions uses **Tcl**'s trace mechanism to provide to the user a clear interface to binary data arrays and structures. It also provides commands for binary input and output.

1 Introduction

Definition 1 *Term* binary data*, used in this document, means any data of fixed size with null bytes inside. So this data can't be treated as plain string in regular* **Tcl**.

This extension adds the following main features to **Tcl**:

- Allocate special memory which will contain binary data objects.
- Clean mechanism for access to the allocated binary objects through plain **Tcl** variables.
- Define fields inside memory array, this allows definition of "C"-like structures.
- Make input/output of the binary objects.

- Takes care of freeing automatically the memory when objects get unset or out of scope (like any **Tcl** variable).

Trace mechanism of **Tcl** used in the extension for providing direct access to the binary data through **Tcl** variables. This is a bit complicated thing in sense of memory usage, because every binary object has its "string" representation in corresponding variable after it was accessed once.

2 Main concepts

Definition 2 **TclBin**'s binary object *is a special structure which is allocated by* **TclBin**. *It has a pointer to allocated memory and contain additional data allowing access to it from* **Tcl**.

Every binary object has a name associated with it in creation time (see `bin_new` command for more information). This name defines also a **Tcl** array (that is a `object()`). Any access to this array cause access to the binary object associated with it. Reading of some field of the array will cause transformation of the binary data to string representation defined by the field. This representation will be returned as contents of the field. Writing to the field will cause transformation of the string to binary data, which will be placed in the object. Each field of the array gives different string representation of the binary data in the object. Currently the following fields are created:

str – binary data is treated as plain string. Every instance of the "impossible" null character (remember that end of the data defined by its length and not by null character, as opposite with regular Tcl), is replaced by "`\0`" and every instance of '\' is replaced by "`\\`".

<*field*> – accesses field defined by `bin_def`. See below.

<*index*> – binary data treated as sequence of bytes , short integers, long integers, network order unsigned 16 bits numbers, network order unsigned 32 bits numbers, or doubles depending on type of the object (see `bin_new`). *Index* must be integer number. One element referenced by index is accessed and its representation depends on the type. (error checking is done and you can't go out of bounds)

<*index*>,hex – binary data treated as sequence of bytes. *Index* must be integer number. Only the byte referenced by index is accessed and treated as hexadecimal number.

hex – binary data is treated as one big hexadecimal number.

obj – this is read-only field. Object type is returned on access. See `bin_new` for more information.

3 Summary of commands

Command Name	Description	"C" function
bin_new	create a new binary object	Bin_NewCmd()
bin_def	define a new field in object	Bin_DefCmd()
bin_info	returns info about binary object	Bin_InfoCmd()
bin_sizeof	returns object or type size	Bin_SizeofCmd()
bin_move	move pointer object inside father	Bin_MoveCmd()
bin_resize	resize pointer object	Bin_ResizeCmd()
bin_copy	copy object to another	Bin_CopyCmd()
bin_read	read object from file	Bin_ReadCmd()
bin_write	write object to file	Bin_WriteCmd()

4 Descriptions

You shall also look at the individual man pages of each commands. Looking through the distribution examples and tests (in `tests/` directory) should also be instructive.

Name: bin_new

Syntax:
`bin_new` *name type sizeExpr ?father? ?offsetExpr?*

Return Value:
Return the name of the newly created object

Exceptions:
Return with `TCL_ERROR` if unable allocate object or invalid father specification.

Description:
Create a new binary object named *name* of type *type* and size *size*. Name must be unique object name or "#auto" in which case name will be generated automatically. Name always returned as result of the command. Type can be any string and is used to label the kind of objects. However, the following `char` , `short`, `int` , `netshort`, `netint`, `double` , `float` , `complex` , `str` are the built-in types of **TclBin**, each one corresponds to "C" language type, str is a null terminated variable length char array (but it's use is not recommended now that you can use the nelems argument with the *char* type within bin_def to create fixed length strings (char* array)), netshort and netint are respectively 16 and 32 bits **unsigned** numbers, in network order (high byte first). Built-in types are not used directly in `bin_new` but rather as field descriptions (in `bin_def`). Rather, when the *type* parameter is the concatenation of built-in type and "*" it means that the object is an array of the base type (e.g. "int*"

which means an array of ints). When binary object A created as array, access to each element is allowed from **Tcl** by `A(index)` syntax, where `A(0)` represents the first element in the array.

SizeExpr must be an expression resulting in an integer specifying the size in BYTES.

If no *father* parameter is specified a new memory chunk is allocated for the object information structure and the data chunk specified by sizeExpr, initially the data part contains garbage.

If *father* is specified it must be name of already created binary object. In this case no memory is allocated for the object data (only a new internal **TclBin** information structure is allocated), but the new object becomes a *pointer object* and it points inside its "father". This feature enables definition of any level of complex data structures inside objects. See also `bin_def` command.

If *offsetExpr* parameter is specified it must be expression which returns integer value. In this case the pointer object will point to *offsetExpr* bytes after the start of the father object. The child object must be fully contained in the father object, otherwise (offset+size is too large) `bin_new` will exit with `TCL_ERROR`.

Name: bin_def

Syntax:
`bin_def` *obj_name field type offsetExpr ?nelems?*

Return Value:
Return "obj_name(field)" string

Exceptions:
Return with `TCL_ERROR` if given not valid object or field can't be placed in the object.

Description:
Defines additional field in binary object given by *obj_name*. Field be named by *field* parameter. It is actually pointer object (see `bin_new` above). It can be accessed by "obj_name(field)". If "nelems" is given, the field is an array of the given type (note that due to **Tcl** limitation you can't use those arrays directly from tcl (like in $obj(array)($index)) except for char arrays which are dealt specially by **TclBin** so accessing obj(string) will work as fixed length string).

Name: bin_info

Syntax:
`bin_info` *?-type? name*

Return Value:
Return info keyed list described below.

Exceptions:
Return with `TCL_ERROR` if given object is not valid.

Description:
This command checks existence of the binary object named by *name* and returns keyed list containing 8 elements (or just the type if the *-type* flag is given). As it is a keyed list, each element is itself list of two values. First value is the key of the element and the second its actual value. Currently the list contains the following pairs:

"data" with pointer to object's data.

"size" with object's size.

"father" with pointer to object's father.

"type" with object's type.

"elemsize" with size of one object's unit (in case of array size of one element)

"deleteproc" with pointer to delete procedure.

"peekpokeproc" with pointer to procedure which converts data for object.

"traceproc" with pointer to trace procedure.

This correspond exactly to a dump of the internal **TclBin** information structure, and only the size,type and elemsize should be of any interest beside debugging purpose...

Name: bin_sizeof

Syntax:
`bin_sizeof` *name*

Return Value:
Return the size in bytes of given object or type (integer).

Exceptions:
Return with `TCL_ERROR` if given object or type is not valid.

Description:
This command receives a *name* which must be valid object or type name. It returns size in bytes.

```
bin_new big buffer 1000;
bin_new small char* 100 big ;
loop i 0 10 {
  # do something on small
  # ...

  # slew to next 100 bytes chunk of big buffer
  bin_move -noerr small;
}
```

Figure 1: `bin_move` example

Name: bin_move

Syntax:
`bin_move` *?-absolute? ?-noerr? object ?offsetExpr? ?sizeExpr?*

Return Value:
Return new offset of the object from its father start

Exceptions:
Return with `TCL_ERROR` if an invalid object is specified or if a move out of father bounds is tried (see also -noerr below).

Description:
This command acts only on pointer objects and on normal objects it returns with `TCL_ERROR`. This command determines new position for the pointer and moves the pointer object inside of its father . This new position determined by offset of the pointer. The offset in bytes determined by the formula offset = offsetExpr ∗ sizeExpr. If *offsetExpr* is omitted it defaults to 1, if *sizeExpr* is omitted it defaults to size of the object. Both offsetExpr and sizeExpr can be integer expressions. When no *-absolute* option specified the offset is added to current pointer's position , otherwise it specified an absolute position from father object byte 0. If *-noerr* option is specified, no exception will be generated when you try to move the object out of father's limits (nothing happens), this is useful to write code like example on Figure 1 which would fail on the last loop without -noerr.

Name: bin_resize

Syntax:
`bin_resize` *object sizeExpr*

Return Value:
Return the new size of the object in bytes

Exceptions:
Return with `TCL_ERROR` if an invalid object was given or if resize would cause the object to be out of father.

Description:
This command acts only on pointer objects and on normal objects it returns with `TCL_ERROR`. This command resizes pointer object to *sizeExpr*.

Name: bin_copy

Syntax:
`bin_copy` *objectDest objectSrc—byteConst ?offsetDestExpr? ?offsetSrcExpr? ?sizeExpr?*

Return Value:
Return number of bytes copied

Exceptions:
Return with `TCL_ERROR` if invalid objects are given.

Description:
This command copies data from *objectSrc* starting from byte *offsetSrcExpr* to *objectDest* starting from byte *offsetDestExpr* . Maximum *sizeExpr* bytes will be copied. If there is no enough place in the destination object for all data, only part of the data will be copied. OffsetDestExpr and offsetSrcExpr defaults to 0 if not specified. SizeExpr defaults to maximum amount of bytes which can be copied without writing out of the destination object. Data is copied using memmove, so areas can overlap (sub pointers objects). If *objectSrc* is an integer instead of an object, bin_copy works like memset and copies the byte value repeateadly over *objectDest*.

Name: bin_read

Syntax:
`bin_read` *?-nobuf? fileId object*

Return Value:
Return number of bytes actually read

Exceptions:
Return with `TCL_ERROR` if invalid file descriptor or object was given or if a read error occurs (number of read bytes before the error is still returned in the error message)

Description:
This command receives a **Tcl** file descriptor *fileid* and object given by *object*. It reads "bin_sizeof object" bytes into the object. When used with Tcl version before 7.5, if *-nobuf* option is specified (or just -) unbuffered read(2) call is used instead of stdio' fread(3) call. With Tcl version 7.5 or higher, Tcl_Channel is used and the *-nobuf* option is not supported.

Name: bin_write

Syntax:
`bin_write` *?-nobuf? fileId object*

Return Value:
Return number of bytes actually written

Exceptions:
Return with `TCL_ERROR` if invalid file descriptor or object was given or if a write error occurs (number of written bytes before the error is still returned as part of the error message)

Description:
This command receives a **Tcl** file descriptor *fileid* and object given by *object*. It writes "bin_sizeof object" bytes from the object. When used with Tcl version before 7.5, if *-nobuf* option is specified (or just -) unbuffered write(2) call is used instead of stdio' fwrite(3) call. With Tcl version 7.5 or higher, Tcl_Channel is used and the *-nobuf* option just implies that Tcl_Flush(3) is called.

----

5 C interface functions

In order to have access to the **TclBin** library in a **Tcl** interpreter the application must call `Bin_Init(interp)` which completely initializes the extension.

To get access to a **TclBin** object from C, the library provides convenient functions like Bin_GetObjectAndCheck which returns the pointer to the object structure or NULL if the name specified is not an object or is not of the specified type (and sets for your the tcl interpreter error message so you just need to return `TCL_ERROR` directly in that case). Example :

```
  Bin_Object *pobj;
  if (!(pobj=Bin_GetObjectAndCheck(interp,argv[1],"double*")))
return TCL_ERROR;
  /* now use pobj->data for data and pobj->size for object size */
```

More informations to be found in the examples source files and in the complete man pages (`cdoc/` directory, starting for instance with `Bin_NewObject(3)`).