Practical Example of Tcl Command Design in a Qt/C++ Graphical Application

Tony Johnson

Mentor Graphics, a Siemen's Company

tony_johnson@mentor.com

Paper Motivation

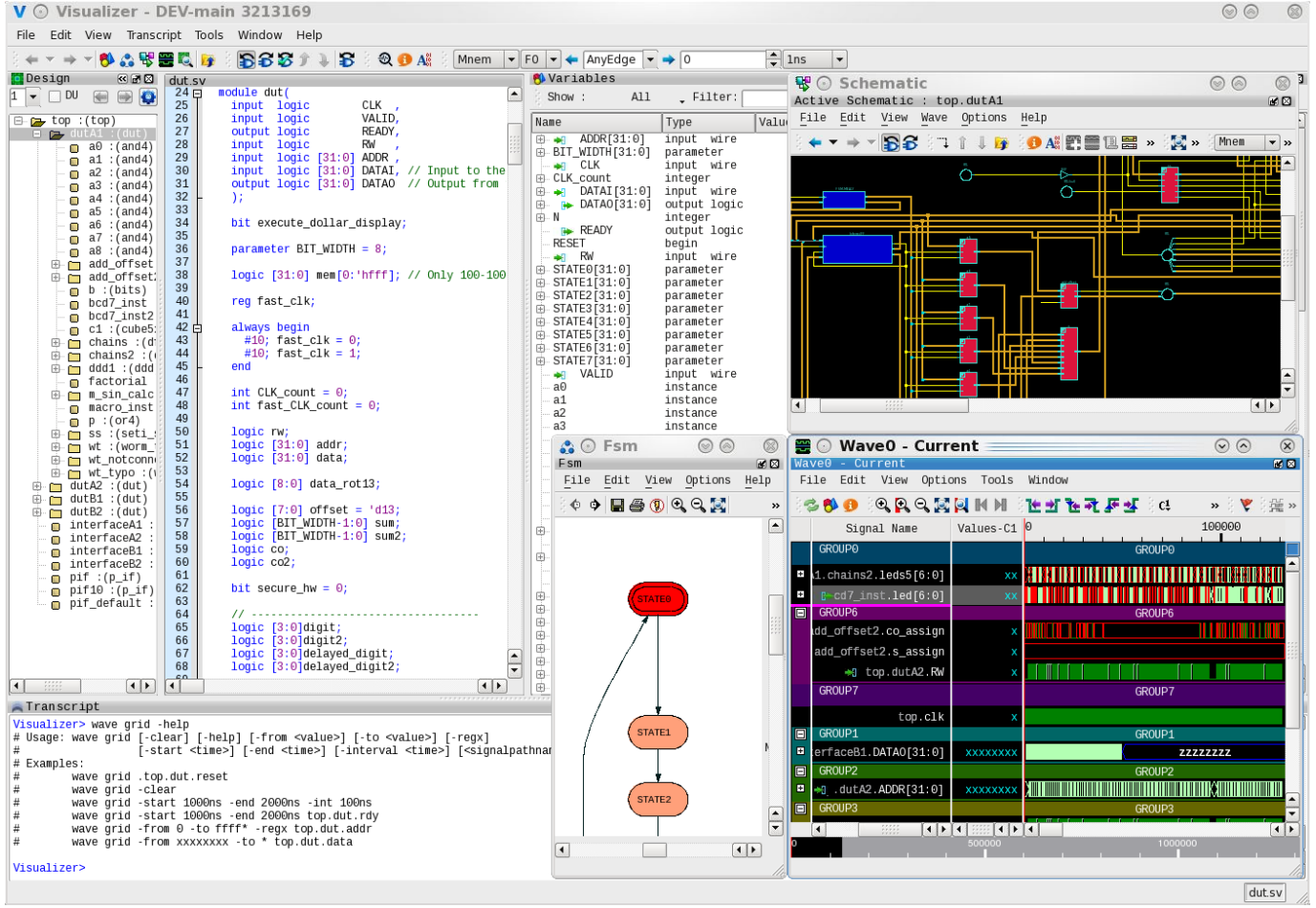
Why write this paper?

- Explain why creating commands to control Graphical Applications is important.
- Provide an example for others to follow in order to create such a command in their GUI.
- Describe why Tcl is a particularly good choice for this purpose regardless of GUI implementation language.
- Offer advice on how to design commands in a way that is user friendly, easily extendable and consistent as they grow over time.

Tcl Command Background and Motivation

Why create a Tcl Command to control a GUI?

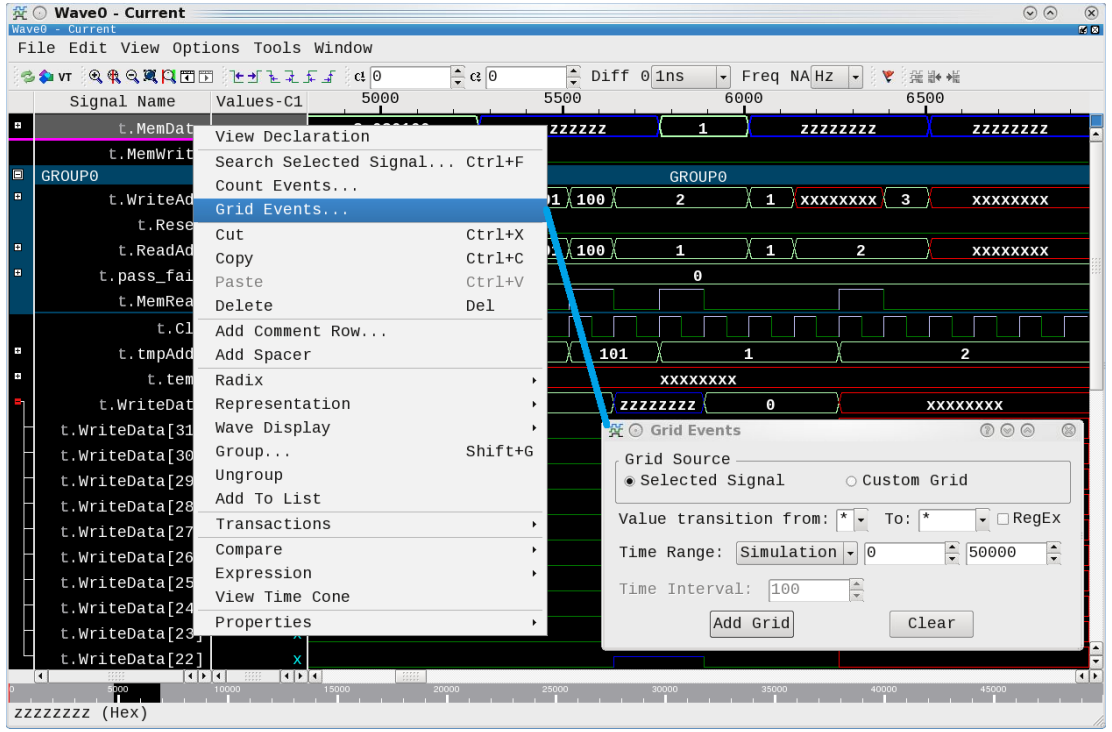
- Testing
- User Control
- 3rd Party Access
- Save/Restore
- Expandability



Brief Wave Window Description

What does the window we will be discussing do?

- What is a Wave window used for?
  - Viewing digital/analog signal activity over time.
- What types of user operations?
  - Cutting/Pasting
  - Adding/Deleting signals/markers
  - Expanding/Collapsing
  - Zooming in/out/full
  - Panning left/right/up/down
  - Creating Grids, Expressions, Comments, Spacers
  - And much more.



Brief Wave Window Description (cont)

What does the "wave" command do?

- wave -help

```
Visualizer> wave -help
# Usage: wave -help                | This message.
#        wave activate             | Activate the specified window or report the active window.
#        wave add                  | Add signals to wave window.
#        wave biometricsearch      | Set or clear Biometric search values.
#        wave cget                 | Returns current configuration value for wave window.
#        wave clear                | Clears the wave window.
#        wave clearselection       | Unselect all
#        wave collapse             | Collapse a particular waveform by index.
#        wave comment              | Add a comment row.
#        wave compare              | Compare two signals highlighting differences.
#        wave concatenate          | Create a concatenation of all selected signals.
#        wave configure            | Query or modify configuration options of the window.
#        wave deleteselected       | Delete the selected objects.
#        wave end                  | Return the index of the last object in the wave window.
#        wave expand               | Expand a particular waveform by index.
#        wave expression           | Create an expression in the wave window.
#        wave find                 | Find the index of the next occurrence of a signal by name.
#        wave get                  | Get the name and/or value of signals in the wave window.
#        wave grid                 | Create a grid in the wave window.
#        wave group                | Create a new group or subgroup.
#        wave grouprename          | Rename a particular group.
#        wave index                | Get the index of selection, insertpoint or end of the wave window.
#        wave insertion            | Set the location of the insertpoint in the wave window.
#        wave list                 | Get the list of currently open wave windows.
#        wave marker               | Create new or modify existing markers in the wave window.
#        wave name                 | Return the name of the active wave window.
#        wave positioncursor       | Control the location of the primary and secondary cursors.
#        wave remove               | Remove (unload) a qwave.db file from Visualizer.
#        wave save                 | Save the wave window format to a file.
#        wave seetime              | Pan the wave window to see a particular simulation time.
#        wave selectall            | Select all objects in the wave window.
#        wave selection            | Set or clear selection of signals by index.
#        wave spacer               | Create a spacer.
#        wave tags                 | Access the qwave.db Tag values (i.e. "F0", "F1", etc).
#        wave timeunit             | Get or set the current time unit of the wave window.
#        wave update               | Enable or disable drawing in the wave window.
#        wave zoomfull             | Zoom full.
#        wave zoomin               | Zoom in.
#        wave zoomout              | Zoom out.
#        wave zoomrange            | Zoom to a particular time range.
```

```
Visualizer> wave grid -help
# Usage: wave grid [-clear] [-help] [-from <value>] [-to <value>] [-regx]
#                  [-start <time>] [-end <time>] [-interval <time>] [<signalpathname>]
# Examples:
#        wave grid .top.dut.reset
#        wave grid -clear
#        wave grid -start 1000ns -end 2000ns -int 100ns
#        wave grid -start 1000ns -end 2000ns top.dut.rdy
#        wave grid -from 0 -to ffff* -regx top.dut.addr
#        wave grid -from xxxxxxxx -to * top.dut.data
```

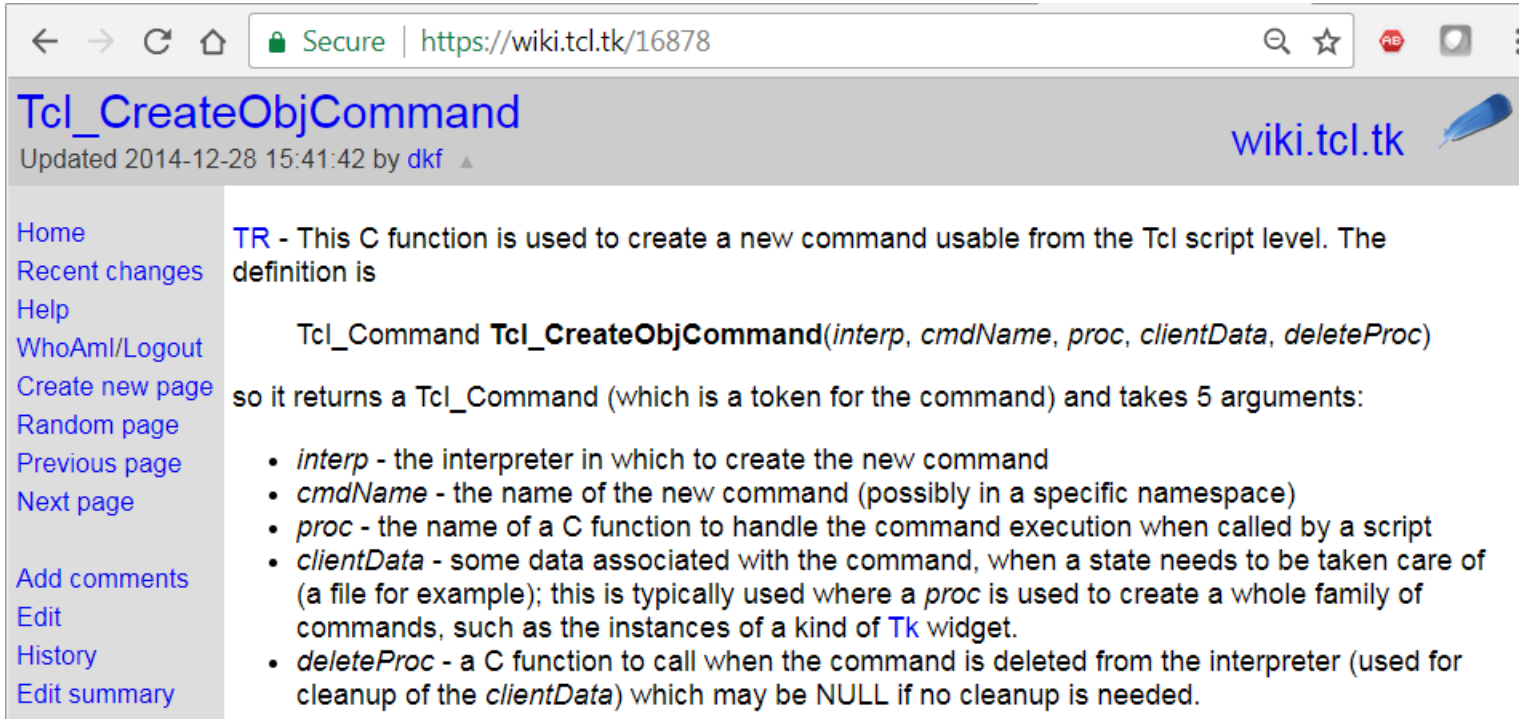
Wave Command Architecture

How do we create such a command assuming we already have a Tcl shell?

- Tcl Command Registration
- String Conversion
- Handle Subcommands
- Provide User Command Help

Tcl Command Registration

How do we connect our Tcl command to our C/C++ application?



Secure | https://wiki.tcl.tk/16878

Tcl_CreateObjCommand

Updated 2014-12-28 15:41:42 by dkf

wiki.tcl.tk

Home
Recent changes
Help
WhoAmI/Logout
Create new page
Random page
Previous page
Next page

Add comments
Edit
History
Edit summary

TR - This C function is used to create a new command usable from the Tcl script level. The definition is

Tcl_Command **Tcl_CreateObjCommand**(*interp*, *cmdName*, *proc*, *clientData*, *deleteProc*)

so it returns a Tcl_Command (which is a token for the command) and takes 5 arguments:

- *interp* - the interpreter in which to create the new command
- *cmdName* - the name of the new command (possibly in a specific namespace)
- *proc* - the name of a C function to handle the command execution when called by a script
- *clientData* - some data associated with the command, when a state needs to be taken care of (a file for example); this is typically used where a *proc* is used to create a whole family of commands, such as the instances of a kind of Tk widget.
- *deleteProc* - a C function to call when the command is deleted from the interpreter (used for cleanup of the *clientData*) which may be NULL if no cleanup is needed.

- **Tcl_CreateObjCommand**(xInterp, "wave", gTclWave, 0, 0);
- int **gTclWave**(ClientData xClientData, Tcl_Interp *xInterp, int xObjc, Tcl_Obj *const xObjv[])

String Conversion

How do we pass strings back and forth between Tcl and Qt/C++

- Tcl_GetString, tclObjCls, QString.toLocal8Bit(), QString.fromUtf8()

  The arguments passed to our “gTclWave” C function are passed as a Tcl_Obj [8] array.
  We used the following strategy to convert these back and forth:

  Tcl to QString:

  ```
  QString lGroupArg;
  lGroupArg.sprintf("%s", Tcl_GetString(xObjv[iii]));
  ```

  QString to Tcl:

  ```
  tclObjCls lResult;
  tclObjCls lBufObj(qPrintable(lWin->mGetName()));
  lResult.mLappend(lBufObj);
  ```

  We defined “tclObjCls” as a class that contains the Tcl_Obj* (as “dObj”) along with convenience methods for creating, printing, reference counting, etc Tcl_Obj objects. For example the mLappend() is defined to be:

  ```
  mLappend(tclObjCls &xObj) {Tcl_ListObjAppendElement(NULL, dObj, xObj.dObj);
  ```

  The function “qPrintable” is defined to be one of the many ways to get string data out of a QString.

  ```
  #define qPrintable(string) (string).toLocal8Bit().constData()
  ```

  As noted in the Future Work section of the paper, instead of toLocal8Bit we should be using the “fromUtf8()” and “toUtf8()” methods for string conversion to more safely handle all possible characters.

Handle Subcommands

How do we keep our subcommands consistent and extensible?

- From https://www.tcl.tk/man/tcl8.4/TclLib/GetIndex.htm:

Tcl_GetIndexFromObj, Tcl_GetIndexFromObjStruct - lookup string in table of keywords

#include <tcl.h>
int **Tcl_GetIndexFromObj**(*interp, objPtr, tablePtr, msg, flags, indexPtr*)
int **Tcl_GetIndexFromObjStruct**(*interp, objPtr, structTablePtr, offset, msg, flags, indexPtr*)

Tcl_Interp ***interp** (in)
Interpreter to use for error reporting; if NULL, then no message is provided on errors.
Tcl_Obj ***objPtr** (in/out)
The string value of this object is used to search through *tablePtr*. The internal representation is modified to hold the index of the matching table entry.
CONST char ****tablePtr** (in)
An array of null-terminated strings. The end of the array is marked by a NULL string pointer. Note that references to the *tablePtr* may be retained in the internal representation of *objPtr*, so this should represent the address of a statically-allocated array.
CONST VOID ***structTablePtr** (in)
An array of arbitrary type, typically some **struct** type. The first member of the structure must be a null-terminated string. The size of the structure is given by *offset*. Note that references to the *structTablePtr* may be retained in the internal representation of *objPtr*, so this should represent the address of a statically-allocated array of structures.
int **offset** (in)
The offset to add to structTablePtr to get to the next entry. The end of the array is marked by a NULL string pointer.
CONST char ***msg** (in)
Null-terminated string describing what is being looked up, such as option. This string is included in error messages.
int **flags (in)**
OR-ed combination of bits providing additional information for operation. The only bit that is currently defined is **TCL_EXACT**.
int ***indexPtr** (out)
The index of the string in *.tablePtr* that matches the value of *objPtr* is returned here.

Handle Subcommands (cont)

What are the benefits to using Tcl_GetIndexFromObjStruct?

```
static optionWaveTable optionWaveCmds[] = {
//optionName optionEnum  optionDesc                optionFlags
{"add",     CMD_ADD,     "Add signals to wave window.", 0},
{"blank",   CMD_BLANK,   "",                            DEP},
{"clear",   CMD_CLEAR,   "Clears the wave window.",     0},
{"comment", CMD_COMMENT, "Add a comment row.",          0},
{"cursor",  CMD_CURSOR,  "Control the cursors.",        TBD},
{"InvokeMenu", CMD_INVOKEMENU, "",                      HID},
…
```

- Command creation method helps to ensure help text is also created.
- Command names defined close together helps ensure consistency.
- Defining and using optionEnum makes finding other commands easy.
- The optionFlags field provides a handy way to hide or deprecate commands.
- Can add as many other fields as you want to this structure.

Handle Subcommands (cont)

How did we call and use Tcl_GetIndexFromObjStruct?

```
int gTclWave(ClientData xClientData, Tcl_Interp *xInterp, int
             xObjc, Tcl_Obj *const xObjv[]) {

    if (Tcl_GetIndexFromObjStruct(xInterp, xObjv[1],
           optionWaveCmds, sizeof(optionWaveTable), "command",
           0, &index) != TCL_OK) {
        return TCL_ERROR;
    } //Else If "-help" passed in for a particular command
        switch((enum optionEnumsWindowCmd)index) {
            case VIS_WAVE_CMD_GRID:
                sWaveGridHelpMsg(xInterp);
                return TCL_OK; break;
            case …
    } //Else call "wave" window command handler passing args
        lActiveWaveWinPtr->mWindowCmd(xInterp, xObjc, xObjv);
```

- Non-existant subcommands automatically handled.
- Top-level Help automatically generated from optionWaveTable.
- Subcommand redirection and help handled by switching on “index”.

Provide Command Help

How do we design the help system to be consistent and extendable?

mWindowCmdGrid() for example:

```
int waveFormWinCls::mWindowCmdGrid(Tcl_Interp *xInterp,
                    int xObjc, Tcl_Obj *const xObjv[]) {
static const char *options[] = {
          "-help", "-clear", "-from", "-to", "-regx",
          "-start", "-end",” -interval", (char*)NULL };

enum wavegridopt {
    WAVEGRID_HELP, WAVEGRID_CLEAR, WAVEGRID_FROM,
    WAVEGRID_TO,   WAVEGRID_REGX,  WAVEGRID_START,
    WAVEGRID_END,  WAVEGRID_INTERVAL };


//Call Tcl_GetIndexFromObj to get “wave grid” subcommand
Tcl_GetIndexFromObj(xInterp, xObjv[iii], options,
                    "option", 0, &lOptIndex)
    switch (lOptIndex) {
        case WAVEGRID_HELP:
            sWaveGridHelpMsg(xInterp);
            return TCL_OK; break;
        case WAVEGRID_CLEAR:  // GUI Clear Method
            mGetWaveFormViewPtr()->mClearGrid();
            return TCL_OK; break;
        case …
```

```
Visualizer> wave -help
# Usage: wave -help              | This message.
#        wave activate           | Activate the specified window or report the active window.
#        wave add                | Add signals to wave window.
#        wave biometricsearch    | Set or clear Biometric search values.
#        wave cget               | Returns current configuration value for wave window.
#        wave clear              | Clears the wave window.
#        wave clearselection     | Unselect all
#        wave collapse           | Collapse a particular waveform by index.
#        wave comment            | Add a comment row.
#        wave compare            | Compare two signals highlighting differences.
#        wave concatenate        | Create a concatenation of all selected signals.
#        wave configure          | Query or modify configuration options of the window.
#        wave deleteselected     | Delete the selected objects.
#        wave end                | Return the index of the last object in the wave window.
#        wave expand             | Expand a particular waveform by index.
#        wave expression         | Create an expression in the wave window.
#        wave find               | Find the index of the next occurrence of a signal by name.
#        wave get                | Get the name and/or value of signals in the wave window.
#        wave grid               | Create a grid in the wave window.
#        wave group              | Create a new group or subgroup.
#        wave grouprename        | Rename a particular group.
#        wave index              | Get the index of selection, insertpoint or end of the wave window.
#        wave insertion          | Set the location of the insertpoint in the wave window.
#        wave list               | Get the list of currently open wave windows.
#        wave marker             | Create new or modify existing markers in the wave window.
#        wave name               | Return the name of the active wave window.
#        wave positioncursor     | Control the location of the primary and secondary cursors.
#        wave remove             | Remove (unload) a qwave.db file from Visualizer.
#        wave save               | Save the wave window format to a file.
#        wave seetime            | Pan the wave window to see a particular simulation time.
#        wave selectall          | Select all objects in the wave window.
#        wave selection          | Set or clear selection of signals by index.
#        wave spacer             | Create a spacer.
#        wave tags               | Access the qwave.db Tag values (i.e. "F0", "F1", etc).
#        wave timeunit           | Get or set the current time unit of the wave window.
#        wave update             | Enable or disable drawing in the wave window.
#        wave zoomfull           | Zoom full.
#        wave zoomin             | Zoom in.
#        wave zoomout            | Zoom out.
#        wave zoomrange          | Zoom to a particular time range.
```

```
Visualizer> wave grid -help
# Usage: wave grid [-clear] [-help] [-from <value>] [-to <value>] [-regx]
#                  [-start <time>] [-end <time>] [-interval <time>] [<signalpathname>]
# Examples:
#        wave grid .top.dut.reset
#        wave grid -clear
#        wave grid -start 1000ns -end 2000ns -int 100ns
#        wave grid -start 1000ns -end 2000ns top.dut.rdy
#        wave grid -from 0 -to ffff* -regx top.dut.addr
#        wave grid -from xxxxxxxx -to * top.dut.data
```

Summary

What lessons have been learned from this work?

- Creating Tcl commands to interact programmatically with a GUI is useful for many reasons.
- Tcl provides excellent resources for implementing these commands.
- The key Tcl routines to remember and leverage are Tcl_CreateObjCommand, Tcl_GetString, Tcl_GetIndexFromObjStruct()
- Designing commands with consistency in mind and with built in help is important.