Graphical Desktop Korn Shell

By George Kraft IV

The Graphical Desktop Korn Shell (DtKsh) is a featured part of the Common Desktop Environment (CDE). DtKsh provides a consistent and reliable graphical Motif shell language that is supported on all CDE-compliant systems.

Portability and pervasiveness are two important characteristics to consider when you are developing code. Using a programming language with a well-defined and stable Application Programming Interface (API) answers the need for portability. A programming language with a large, established installed base provides pervasiveness. Although Perl, Tcl/Tk, Common Gateway Interface (CGI), and Java have large installed bases, they are not suited for some projects. The reason for this is their inconsistent installation base due to their lack of a well-defined API or their rapidly changing API.

The Desktop Korn Shell (`DtKsh`) that comes with the Common Desktop Environment (CDE) is built on the ksh93 standard with X, Xt, Motif®, ToolTalk®, and CDE built-in APIs. Unlike Perl and Tcl/Tk, major vendors have built and supported `DtKsh` through the CDE initiative. Using `DtKsh`, desktop programmers can develop and/or prototype plug-and-play Graphical User Interface (GUI) applications that are compatible on all CDE-compliant systems without compilation. Although `DtKsh` applications are interpreted for portability, they can be easily migrated to Motif in C for performance.

Tcl/Tk can be ported to C with the aid of special Tcl/Tk libraries; however, programmers are as disadvantaged with the C Tcl/Tk libraries as they are with the Tcl/Tk shell, because of a not-so-standard Application Programming Interface. `DtKsh`, unlike Tcl/Tk, provides a well-established API set where the programmer's knowledge transcends from C to shell programming.

Desktop Korn Shell

- Based on Korn Shell
- New desktop built-in APIs
- Shipped with CDE-compliant systems
- Plug and play
- Machine-independent compatibility
- Migration path to C

Figure 1. The advantages of DtKsh

DtKsh Benefits

In AIX, `/bin/ksh` is an XPG4-compliant version of ksh88. CDE's `/usr/dt/bin/dtksh` on AIX is based on the newer ksh93 standard. Ksh93 now includes floating-point mathematics, associative arrays, new string operations, hierarchical variables, reference variables, developer-extendable APIs using attached shared libraries, and character class patterns.

George Kraft IV

Floating-point mathematics. Korn Shell variables can be type cast, or defined, to various aggregate data types. Floating-point mathematics is a new feature to Korn Shell that enables the assignment and operation of decimal values. The following example defines the floating-point variable PI, then assigns to it the decimal value of 3.1459.

```
typeset -F PI # define "PI" as a float
PI = 3.1459
```

Associative arrays. Instead of using positive integer indices, associative arrays allow elements of an array to be addressed using alphanumeric strings. The following example shows `SYSINFO` as an array containing information about an operating system. The associative `SYSINFO` array can be indexed with the alphanumeric string of `os` to find the string value of `AIX`.

```
typeset -A SYSINFO # define "SYSINFO"
                   # as an
                   # associative array
SYSINFO["os"]=AIX
```

New string operations. Six new string operations were introduced in ksh93. These new operations provide substringing and substitution of a string pattern with an alternate. Substringing permits extraction of a smaller string given an offset where to begin and possibly its length.

- A substring of a larger string can be extracted by length at a given starting point, or a substring can be taken by starting at the offset within the larger string and stopping at the end of the string. The following shows a substring of a given length:

  ```
  ${variable:offset:length}
  ```

- A substring of no particular length can be taken by just providing the offset.

  ```
  ${variable:offset}
  ```

String substitution of a character pattern can be performed for the first occurrence, repeated occurrence, at the beginning of the string (prefix), or at the end of the string (suffix).

- Substitute the first occurrence of a pattern with the alternate string:

  ```
  ${variable/pattern/string}
  ```

- Substitute all occurrences of a pattern with the alternate string:

  ```
  ${variable//pattern/string}
  ```

- Substitute the pattern prefix with the alternate string:

  ```
  ${variable/#pattern/string}
  ```

- Substitute the pattern suffix with the alternate string:

  ```
  ${variable/%pattern/string}
  ```

Hierarchical variables. Hierarchical variables, or compound names, enable C structure-like aggregate data types. This allows Korn Shell to store information in variables in an associative fashion. For example, if we had a box with the width of

```
#! /usr/dt/bin/dtksh

  main()
  {
    XtInitialize TOPLEVEL dtHello DtHello "$@"

    XmCreateMessageDialog HELLO $TOPLEVEL hello \
            dialogTitle:"DtHello" \
            messageString:"$(print "Hello\nWorld")"

    XmMessageBoxGetChild HELP $HELLO DIALOG_HELP_BUTTON
    XtUnmanageChild $HELP
    XmMessageBoxGetChild CANCEL $HELLO DIALOG_CANCEL_BUTTON
    XtUnmanageChild $CANCEL

    XtAddCallback $HELLO okCallback exit
    XtManageChild $HELLO
    XtMainLoop

   }

   main
```

Figure 2. DtKsh "Hello World" program

80 and height of 24, then we could store all that information in one hierarchical variable instead of separate and disjointed variables of storage. Each element of the compound name must be used before setting submembers.

```
BOX= # declare before assigning submembers
BOX.WIDTH = 80
BOX.HEIGHT = 24
```

Reference variables. Referencing allows a variable to point to the same value as another variable; both variables reference the same value as shown below:

```
# name reference
typeset -n FOO=BAR

FOO="Hello World"

# print "Hello World"
print ${BAR}
```

Desktop built-in commands. Korn Shell provides some standard X, Xt, Motif, POSIX® internationalization, and CDE C language APIs directly built into the shell. Direct access to these APIs from the shell provides a significant runtime performance improvement for `DtKsh` shell applications. Using the standard X and Motif APIs, with some semantic changes to the source, makes it possible for `DtKsh` shell scripts to be migrated to C and compiled.

POSIX internationalization. Korn Shell provides the shell equivalent of the C language POSIX internationalization APIs `catopen` and `catgets`. The internationalization APIs allow the shell program to change its message catalog depending on its language. Internationalized shell scripts enable multilingual support.

Character class patterns. Regular expressions in the shell are enhanced by predefining a set of character class patterns. Now we can easily match certain classes of characters by using the `[[:class:]]` notation where as class is defined as `alnu`, `alpha`, `cntrl`, `digit`, `graph`, `lower`, `upper`, `print`, `punct`, `space`, and `xdigit` as a specified class.

```
#include <stdlib.h>

#include <Xm/MessageB.h>

main(argc, argv)
        int argc;
        char **argv;

{

       Widget topLevelShell, hello, help, cancel;
       Arg xargs[10];
       int n;
       XmString title, greet;

topLevelShell = XtInitialize(argv[0], "DtHello",
       NULL, 0, &argc, argv);

hello = XmCreateMessageDialog(topLevelShell, "hello",
NULL, 0);

title = XmStringCreateSimple("DtHello");
greet = XmStringCreateLtoR("Hello\nWorld");

XtVaSetValues(hello,
      XmNdialogTitle,title,
      XmNmessageString,greet,
      NULL);

XmStringFree(title);
XmStringFree(greet);

help = XmMessageBoxGetChild(hello, XmDIALOG_HELP_BUTTON);

XtUnmanageChild(help);
cancel = XmMessageBoxGetChild(hello,
XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild(cancel);

XtAddCallback(hello, XmNokCallback, exit, NULL);

XtManageChild(hello);

XtMainLoop();

}
```

Figure 3. C language "Hello World" program

```
# only print files that
# begin in upper case
print  [[:upper:]]*

# old way
print [A-Z]*
```

DtKsh 'Hello World' Source

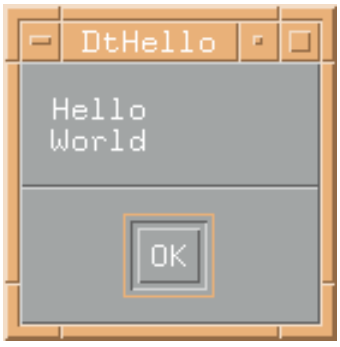
The familiar “Hello World” Motif application, shown in Figure 2, is actually written in `DtKsh` instead of C. Similar to C, we initialize the top-level shell widget, then start building the GUI application. Figure 2 shows a standard Motif message dialog using the familiar `XmCreateMessageDialog` API.

In `DtKsh`, handles to widgets can be retrieved, widgets can be managed and unmanaged, and callbacks can be created. Afterwards, the program enters into the Xt Intrinsic's main loop via `XtMainLoop` where it processes X protocol events. In this case, clicking on the “OK” button would be an event processed by the event loop.

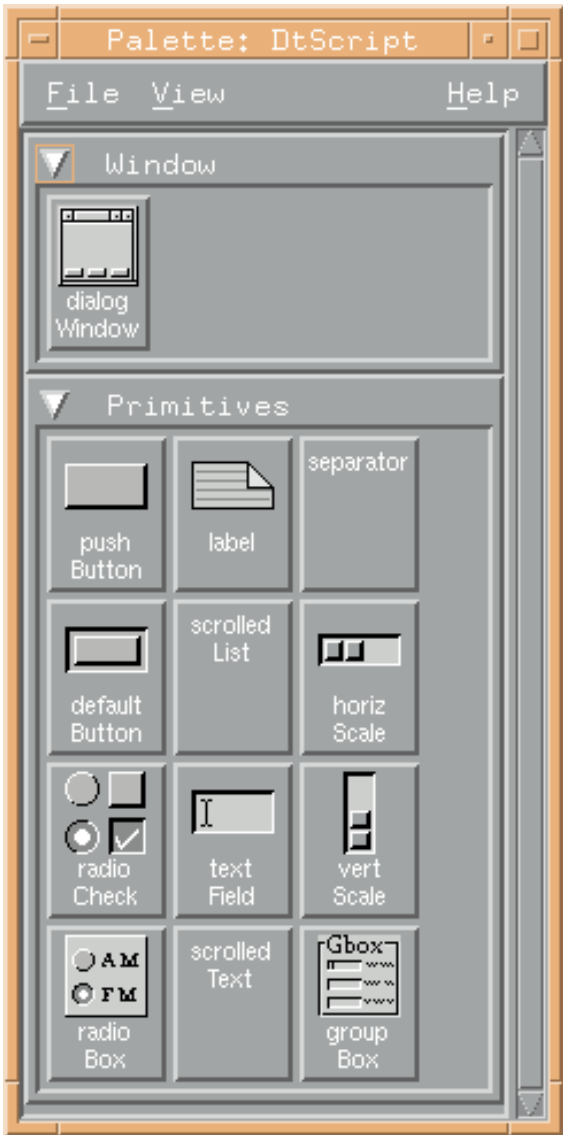
Motif “Hello World” Source

The Motif “Hello World” `DtKsh` application in Figure 2 can be easily ported to C with a few minor changes, shown in Figure 3. By adding some include files, defining some variables, adding some commas and semicolons, and sprucing up some arguments, we have a C program. The result is that `DtKsh` shell scripts make the same API calls as the C Motif application.

AIX provides some extra `DtKsh` help through a GUI builder. Developers can drag and drop widgets onto a canvas, then add some logic code to enable the application to do some work. Like any



DtKsh and Motif "Hello World" application



AIX's DtScript GUI Builder

GUI builder, the code is somewhat verbose; however, it is consistent and portable. AIX is the only version of UNIX that offers this feature.

User Extendable

Developers can create their own new APIs for `DtKsh` by creating glue layer libraries. Glue layer libraries enable `DtKsh` to be extended with built-ins for functions such as system management and networking. The performance advantage of using built-in functions rather than calling to an external command is that built-ins execute within the process of the shell script. Commands that are called externally must

create new resources in the operating system and run as separate processes.

DtKsh glue layer libraries pass arguments between a normal UNIX C library and the DtKsh shell, and they return a success or failure status. The following list provides a few rules for creating a glue layer:

- Name the function with a b_ prefix
- Function returns a 0 integer for success, or between 1 and 255 for failure
- Function should take argc and argv as input
- Link your glue layer libraries shared

```
#!/usr/dt/bin/dtksh

builtin -f ./libexample.o example # dynamically load shared
library

example hello world # call newly loaded builtin command
```

Figure 4. Example of DtKsh

```
int b_example(int argc, char *argv[])
{

      int i;

      for (i = 0; i < argc; i++) {
            printf("argv[%d] = %s\n", i, argv[i]);
      }

      return(0);

}
```

Figure 5. Example of C to shell glue

Figure 4 shows a DtKsh shell script that dynamically loads the "example" shared glue layer library. Once the glue layer library is loaded and the new built-in APIs are defined, then the script can make direct calls with arguments to the new built-in functions. In Figure 4, the example built-in is called with the hello world arguments.

By providing inline built-in functions, we can run scripts much faster because we are not relying on outside programs running as separate system processes. Figure 5 shows the C glue layer for the example built-in shared library. Following the rules outlined above, we prefix the example function with a b_, and we pass in an argument vector and its size. Then after the function has done its work, we return 0 for success and a positive integer for failure. DtKsh built-in functions can also act as procedures that pass environment variables in and out through its argument list. See *Desktop Korn Shell Graphical Programming*[1] by J. Stephen Pendergrast, Jr. (ISBN 0-201-63375-2) for more details on how to pass in and get back environment variables from built-in procedures.

Conclusion

We have seen that the Desktop Graphical Korn Shell provides programmers with the standard ksh93 baseline APIs with the addition of the X Window System®, Motif, and the Common Desktop Environment. Shell programmers can write portable shell scripts, prototype GUI shell scripts, and migrate GUI shell scripts to faster running C programs. DtKsh also provides programmers with the ability to extend the shell language with built-in shared libraries so that scripts can benefit from feature-rich libraries, such as those for configuration management.

George Kraft IV, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Kraft is an advisory software engineer for IBM's new Network Computer Division. He recently moved from IBM's RS/6000 Division where he worked on the AIX integration of the IBM Network Station. He has a BS in Computer Science and Mathematics from Purdue University.

[1] Also see the following Web site: http://www.aw.com/cp/pendergrast.html.