Command Line Parsing in VisionX V4

Like many features in C the basic programing tools for passing user specified parameters to a program are very primitive. By using a command line parsing function such as VXparse the programming and user friendliness of command line arguments can be greatly improved.

There are many different ways of defining the command line syntax for a program; two of the most popular are: position dependent and required prefixes. For example a position dependent syntax may expressed as follows:

```
prog1  <arg1> <arg2> [options]
```

For prog1 there must be two parameters specified as strings in the correct order followed by any program options. Options are usually specified by a string with the first character being a "-". As an example of the above syntax one might write

```
prog1 infile outfile –t –fix
```

In the prefix style all value parameters (file-names and numeric values) have a specific prefix; consider:

```
prog2 if=<arg1> of=<arg2> [options]
```

For this style there is much more flexibility in how commands may be specified, with the additional cost of needing to include the prefixes; for example, all of the following statements re equivalent

```
prog2 if=infile of=outfile –t –fix
prog2 –fix of=outfile –t if=infile
prog2 –t –fix of=outfile if=infile
```

Typically command line parameters are passed to a C program as a set of character strings and a basic code for prog1 may be written as follows:

```
#include <stdio.h>
#define ARG1  0;
#define ARG2  1;
#define OPT1  2;
#define OPT2  3;
int main(int argc, char** argv)
{
char* arg1,  *arg2;
int  optiont, optionfix;
arg1 = argv[ARG1];
arg2 = argv[ARG2];
/* program section here involving argc, argv[OPT1] and argv[OPT2] which will
   determine which of the option variables option and optionfix need to be set    */
 ...
```

The VXparse program provides parsing of the command line and matching to program parameters has two advantages. First, a command line syntax is implemented that can be used with both prefix and positional parameter schemes. Second, program documentation is conveniently supported for both the program developer and the program user.

```
#include "VisXV4.h"
#include "Vutil.h"

VXparam_t par[] = {      /*  command line structure                */
{   "if=",    0,         " prog2: input file name                  "},
{   "of=",    0,         " output file name                        "},
{   "–t",     0,         " Option to select text output            "},
{   "–fix",   0,         " Option to convert output to integers    "},
{   0,        0,         0}   /* list terminator */
};
#define  IVAL   par[0].val   /* these defines give symbolic names   */
#define  OVAL   par[1].val   /* to the second element of the par    */
#define  OPTT   par[2].val   /* structure. For example, IVAL is the */
#define  OPTFIX par[3].val   /* string matched to "if=" (if it is   */
                             /*  found)                         `   */
main(argc, argv)
int argc;
char *argv[];
{
int  optiont, optionfix;
arg1 = IVAL;
arg2 = OVAL;
optiont = OPTT ? 1 : 0;
optionfix = OPTFIX ? 1 : 0;
```

The above program fully supports the prefix syntax of prog2; it also supports the syntax of prog1. In terms of program documentation to function of each command line parameter is describer in the third field of the VXparam_t struct. Also, if a single "-" argument is given to the program these character strings will be printed to the terminal as an interactive guide to the user.

The following source code is a program that illustrates the use of VXparse to parse command line parameters; including how to convert numeric string values to program variables and the use of option parameters.

```
/*****************************************************************/
/* Example program vpex to demonstrates the use of VXparse      */
/*****************************************************************/
#include "VisXV4.h"
#include "Vutil.h"
```

```
VXparam_t par[] = {      /*  command line structure                */
{   "if=",    0,         " input file name                         "},
{   "of=",    0,         " ouput file name                         "},
{   "-i",     0,         " a simple flag                           "},
{   "n=",     0,         " an integer input value                  "},
{   "s=",     0,         " a floating point input value            "},
{   "xy=",    0,         " input one or two values                 "},
{   0,        0,         0}   /* list terminator */
};
#define  IVAL   par[0].val   /* these defines give symbolic names   */
#define  OVAL   par[1].val   /* to the second element of the par    */
#define  IFLAG  par[2].val   /* structure. For example, the string  */
#define  NVAL   par[3].val   /* matched to "if=" (if it is found)   */
#define  SVAL   par[4].val   /* may be referred to in the program   */
#define  XYVAL  par[5].val   /* as IVAL                             */
extern double atof();

int main(int argc, char** argv)
{
    int n;
    float s;
    int x, y;

    VXparse(&argc, &argv, par);  /* parse the command line */

    if (IVAL)
        (void) fprintf(stdout,  "input file name is %s\n", IVAL);
    if (OVAL)
        (void) fprintf(stdout, "output file name is %s\n", OVAL);
    if (IFLAG)
        (void) fprintf(stdout, "Option -i has been selected\n");
    n = 0;    /* default for n */
    if (NVAL) {
        n = atoi(NVAL);
        (void) fprintf(stdout, "n= (integer argument) is %d\n", n);
    }
    s = 0.0;  /* default for s */
    if (SVAL) {
        s = (float)atof(SVAL);
        (void) fprintf(stdout, "s= (float argument) is %f\n", s);
    }
    x = y = 0; /* defaults for x and y */
    if (XYVAL) /* input two values     */
        switch (sscanf(XYVAL, "%d,%d", &x, &y)) {
        case 2:
            break;
        case 1:
            y = x;
            break;
        default:
            (void) fprintf(stderr, "Bad input to xy=\n");
            exit(1);
    }
    exit(0);
}
```

Example Execution of the VXparse Program vpex

```
> vpex -
Usage: vpex [-H] [-] [-help]
 [if=]  input file name
 [of=]  ouput file name
 [-i ]  a simple flag
 [n= ]  an integer input value
 [s= ]  a floating point input value
 [xy=]  input one or two integers

> vpex if=file1 of=file2
input file name is file1
output file name is file2

> vpex n=3 file1
input file name is file1
n= (integer argument) is 3

> vpex -i xy=2 n=1 -o file2
output file name is file2
Option -i has been selected
n= (integer argument) is 1
xy= (argument pair) 2 2

> vpex -H
Usage: vpex [-H] [if=<inputfile>] [of=<outputfile>] [-i] [n=<value>]
      [s=<value>] [xy=<value>]

> vpex file1 -o file2 3 8 6
input file name is file1
output file name is file2
n= (integer argument) is 3
s= (float argument) is 8.000000
xy= (argument pair) 6 6

> vpex 3 8 6
input file name is 3
n= (integer argument) is 8
s= (float argument) is 6.000000

> vpex if= of=
input file name is
output file name is

> vpex n= s= xy=
n= (integer argument) is 0
s= (float argument) is 0.000000
xy= (argument pair) 0 0
```