VisionX V4 Programmers Guide

Anthony P. Reeves
School of Electrical Engineering
Cornell University

August 13, 1998

1 Compiling VisionX Programs

The VisionX command `vcc` should be used to compile all VisionX programs. This command will invoke the compiler and link in all the libraries necessary for the VisionX programming tools. For example, the command to compile the program vpex.c, given in Appendix A, is:

```
vcc vpex.c -o vpex
```

For further information see the man page for vcc(1). This documentation may be accessed on the computer system by typing the command:

```
man vcc
```

2 VisionX Programming Tools

VisionX provides a library of C functions that perform most of the common operations involved in writing image processing modules. These functions perform operations such as command line parsing, file manipulation, and memory allocation. Examples of the use of the VisionX Programming Tools are found in *Appendices A and B*.

VisionX data files are constructed as a sequence of *elements* each element is a data structure that has three attributes: a type which identifies the the class of the elements data, a size which gives the size of the elements data in bytes and the data itself.

When a VisionX file is read it is loaded into memory as a doubly linked *list* of elements. Similarly, the programmer, using VisionX programming tools typically creates a list of VisionX elements which are then written to a VisionX output file with a single VisionX write command.

Command Line Parsing

VisionX supports a powerful convention for passing arguments to a program through the function `VXparse`. An example of the use of `VXparse` is found in *Appendix A*. Further information may be found in the man page VXparse(3).

VisionX File Access Functions

New programs developed with VisionX should make use of a new set of V4 VisionX tools referred to as the *File Access* tools. These functions make maximum use of data encapsulation to shelter the programmer from the overhead and details of VisionX file manipulation; see the VisionX section 3 man pages (especially VXopen(3), VXread(3), and VXmakeimage(3)) for details. The use of these tools is discussed in the following sections.

3 Programming with VisionX File Functions

VisionX programs that are developed with the structured VisionX programming tools usually have the following structure:

1. Declare global variables and VisionX file structures
2. Define the prefixes for the command line parameters
3. Start main program: parse command line and open files
4. Main program body
5. Close data files and exit

A basic VisionX program that illustrates all the main features of the structured VisionX programming tools is shown in Figure 1. Since this structure is similar for most VisionX programs, the example program may be used as a template for developing other programs.

Any VisionX program should be able to deal with any VisionX file, and should accept the flexible VisionX command line structure. The following code walk through illustrates the five stage framework that is used in all structured VisionX programs.

1. Global declarations

Specify the VisionX include file and declare VisionX file structures.
The VisXV4.h file is required for every VisionX program Vutil.h is an optional file that contains a number of useful macros and various standard include files such as stdio.h, math.h and string(s).h.

```
#include "VisXV4.h"     /*  VisionX structure include file    */
#include "Vutil.h"      /* useful macros, stdio.h etc.        */
                        /*                                    */
VisXfile_t *VXin,       /* input file structure               */
           *VXout;      /* output file structure              */
VisXelem_t *vxlist;     /* VisX data structure                */
```

In general, every file is associated with a file structure “VisXfile_t *”, also a data list “VisXelem_t *” is used to hold the data read from or to be written to files.

2. Command line parameters

Define the prefixes for the command line parameters.
The par structure contains a prefix, a value and a brief description for each command line parameter. Both positional (without prefix) and prefix spefication of command line parameters is supported. By convention ”if=” specifies the primary input file and ”of=” specifies the primary output file.

```
/************************************************************/
/* Example VisX4 program  vcp --  Copy a VisX data file     */
/* Syntax:                                                  */
/*        vcp [-f] if=infile of=outfile                     */
/************************************************************/

#include "VisXV4.h"       /* VisX structure include file    */
#include "Vutil.h"        /* useful macros, stdio.h etc.    */
                          /*                                */
VisXfile_t *VXin,         /* input file structure           */
           *VXout;        /* output file structure          */
VisXelem_t *vxlist;       /* VisX data structure            */
                          /*                                */
VXparam_t par[] =         /* command line structure         */
{
{   "if=",    0,          "input file  vcp: file copy"},
{   "of=",    0,          "output file"},
{   "-f",     0,          "frame-i/o option"},
{    0,       0,          0}, /* list termination */
};

#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  FFLAG  par[2].val

main(argc, argv)
int argc;
char *argv[];
{
    VXparse(&argc, &argv, par);    /* parse the command line    */
    VXin  = VXopen(IVAL, 0);       /* open input file           */
    VXout = VXopen(OVAL, 1);       /* open the output file      */

    if(!FFLAG){    /* do complete file i/o */

         vxlist = VXread(VXin);    /* read data                 */
         VXwrite(VXout, vxlist);   /* write data                */

    }else {        /* do frame i/o */
         while ((vxlist = VXreadframe(VXin)) != VXNIL){
                                   /* for each frame read     */
             VXfupdate(VXout, VXin);     /* update global constants */
             VXwriteframe(VXout,vxlist); /* write frame             */
             VXdellist(vxlist);          /* delete the frame        */
         }
    }
    VXclose(VXin);                 /* close input file          */
    VXclose(VXout);                /* close output file         */
    exit(0);
}
```

Figure 1: Example VisionX program

```
VXparam_t par[] =         /* command line structure      */
{
{   "if=",    0,          "input file  vcp: file copy"},
{   "of=",    0,          "output file"},
{   "-f",     0,          "frame-i/o option"},
{    0,       0,          0}, /* list termination */
};

#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  FFLAG  par[2].val
```

Each prefix is specified by two statements. For example, the string matched to the ”of=” command parameter is referred to by OVAL in the program. The VXparse subroutine parses the command line and sets the .val elements of the par data structure to the text strings associated with the matched parameters. If the user does not specify the ”of=” parameter then OVAL is set to a NULL string by VXparse.

3. Main program initialization

Parse the command line and open files.
The second argument to VXopen specifies if a file is to be opened for reading(0) or writing (1).

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

    VXparse(&argc, &argv, par);   /* parse the command line   */
    VXin  = VXopen(IVAL, 0);      /* open input file          */
    VXout = VXopen(OVAL, 1);      /* open the output file     */
```

4. Main program body

There are two modes of processing VisionX files. Either the whole file may be read into a list structure or file ”frames” may be read one at a time. The vcp program illustrates both of these modes. For complete file i/o the structure is:

```
     vxlist = VXread(VXin);    /* read data                  */

          /* do any modifications to vxlist or create     */
          /* a separate list for output                   */

     VXwrite(VXout, vxlist);   /* write data                 */
```

To process the data file one frame at a time the structure is:

```
    while ((vxlist = VXreadframe(VXin)) != VXNIL){
                                /* for each frame read     */
        VXfupdate(VXout, VXin);   /* update global constants */
          /* do any modifications to vxlist or create   */
          /* a separate list for output                 */
        VXwriteframe(VXout,vxlist); /* write frame          */
        VXdellist(vxlist);          /* delete the frame     */
    }
```

In this case a new vxlist is created with each call to VXreadframe. Any global constants that are to be used for all frames are maintained in a list VXflist(VXout). The VXupdate command ensures that this list is maintained with any new global information that is extracted from the input file with the call to VXreadframe.

5. File closing and exit

Any well written program should close all opened files and exit with exit(0).

```
    VXclose(VXin);              /* close input file          */
    VXclose(VXout);             /* close output file         */
    exit(0);
}
```

4 Image Processing VisionX V4 Programs

A number of tools are available to simplify the development of VisionX programs for image processing. An image is considered as a two dimensional array of pixels. In a VisionX file an image is represented by two elements: a bounding box which defines the image structure and a pixel vector which contains the data. A number of different base types for pixels are supported. In addition, each image may have multiple channels. For example, in a color image each pixel has three components (typically red, green and blue).

Within a program a the image programming tools create and maintain a data structure of type `VisXimage_t` to aid accessing the pixels in an image.

An image structure (including the two list elements) may be dynamically created in VisionX with the `VXmakeimage` function. The syntax for this function is:

```
VXmakeimage( &<i-structure>,<type>,<bounding box>,<channels>);
```

Where `i-structure` is the structure to be initialized, `type` is the base type for the pixels, `bounding box` specifies the bounding box (in floating point numbers) and `channels` specifies the number of elements for each pixel.

For example, consider the following image declaration:

```
VisXimage_t foo;
   VXmakeimage ( &foo, VX_PFLOAT, (float){0, 5, 0, 8}, 3);
```

A (5 x 8) pixel image is created in which each pixel consists of three floating point numbers. The bounding box is specified by an array of four floating point numbers (xmin, xmax, ymin, ymax) and the number of channels is specified by an integer.

Pixel types:

The base pixel types supported by VisionX and their VisionX names are given in the table below:

Name	C-Type	Ext	Description
VX_PBYTE	unsigned char	.u	most usual byte format
VX_PFLOAT	float char	.f	32-bit floating point
VX_PDOUBLE	double	.d	64-bit floating point
VX_PBIT	unsigned char	.b	1-bit (packed 8/byte)
VX_PCHAR	char	.c	8-bit signed integer
VX_PSHORT	short	.s	16-bit signed integer
VX_PINT	int	.i	32-bit signed integer
VX_PIDX	unsigned char	.u	8-bit color index

The Ext field specifies the element of the VisXimage data structure which will point to the two dimensional array.

Data Access:

VXmakeimage allocates memory for the image and creates a two dimensional array structure to conveniently access this structure. The following four structure elements are set by VXmakeimage:

```
.xlo  the lowest defined x index.
.xhi  the highest defined x index
.ylo  the lowest defined y index
.yhi  the highest defined y index
```

In addition, the extension corresponding to the pixel data type points to the two dimensional array.

For example, consider the previous (5 x 8) image declaration: All pixels in the image may be set to zero with the following program sequence:

```
{int i, j;
   for ( i = foo.ylo; y <= foo.yhi; i++ )
       for ( j = foo.xlo; x <= foo.xhi; j++ )
           foo.f[i][j] = 0.0;
}
```

The array must be referred to as foo.f since it was declared to be of type VX_PFLOAT. The values for the index limits for this example are as follows: foo.xlo = 0, foo.xhi = 14, foo.ylo = 0, and foo.yhi = 7. The x-range is three times the size of the bounding box since each pixel requires three consecutive floating point numbers. That is, the three components of the first pixel in the array are: foo.f[0][0], foo.f[0][1] and foo.f[0][2].

Writing a File:

The VXmakeimage function generates a VisionX list structure containing a bounding box element an image element; for multichannel images a channel specification element is also created. The list is located in the structure element foo.list; therefore the following program sequence could be used to write a file that contains our example image:

```
{ VisXfile_t fout;
       fout = VXopen("<file name>", 1);
       VXwrite(fout, foo.list);
       VXclose(fout);
}
```

Reading a File:

VisionX files may contain a large number of different types of elements; the function VXsetimage has been developed to construct an VisXimage structure to conveniently access an image if a pixel type element is found in the file. The syntax for this function is:

```
 VXsetimage ( &<i-structure>, <pixel-ptr>, <file-struct> );
```

Where `pixel-ptr` is a pointer to a pixel data element in a VisionX data list, and `file-struct` is the structure returned by the VXopen command.

For example, an image structure could be set up with the contents of an VisionX image file with the following program sequence:

```
VisXfile_t *ifile;
VisXelem_t *list, ptr;
VisXimage_t fie;

       ifile = VXopen("<filename>", 0);
       list  = VXread(ifile);
       if (VXNIL != (ptr = VXfind(list, VX_PBYTE))
           VXsetimage( &fie, ptr, ifile);
```

The function VXfind will locate the first element in the list of type VX_PBYTE (8-bit unsigned pixels); therfore, the given segment will only create a structure if a byte pixel image is located in the data list. (Note: you can use VXfindin for situations where more than one pixel type is possible). In addition to the array accessing structure elements, the following structure elements are set by VXsetimage.

```
.type    the pixel type
.chan    the number of channels
.bbx     the bounding box
.imitem  the item in the list where the image is located
```

Example Program:

A complete VisionX program that illustrates the use of the image structure tools is shown in Figure 2. This program reads a VisionX file and if a byte image is located in that file it generates an output image in which all the pixels have been converted from unsigned char to the short pixel format. Only, the new image is output; any other data items in the input file are ignored.

The simple program presented in Figure 2 will work on simple files which just contain a single byte image. The following more advanced program segment shown in Figure 3 performs the same function but on a VisionX file of arbitrary complexity and length. Files may contain many different types of data, the new program will just replace the *byte* images with *short* images all other data elements will be maintained without modification. The revised program has the following features:

1. The input file may contain multiple images; the file may also be organized in ”frames” which will be read as a unit; files with a large number of images would overflow memory if they were read as a single unit with VXread.

2. The new image replaces the old image in the data list then the list is written. In this way any non byte image information present in the file will be maintained.

3. The image structures must be deallocated by VXresetimage to free memory associated with them before they are reused for the next image. The images in the file may have different dimensions.

```
/*************************************************************/
/* Example VisX4 program vimt                                */
/*            Convert a byte image to a short image          */
/* Syntax:                                                   */
/*        vimt if=infile of=outfile                          */
/*************************************************************/

#include "VisXV4.h"          /* VisX structure include file     */
VisXfile_t *VXin,            /* input file structure            */
           *VXout;           /* output file structure           */
VisXelem_t *VXlist;          /* VisX data structure             */
VisXelem_t *vptr;            /* pixel element pointer           */
VisXparam_t par[] =          /* command line structure          */
{
    "if=",    0,   /* input file  */
    "of=",    0,   /* output file */
     0,       0    /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

main(argc, argv)
int argc;
char *argv[];
{

VisXimage_t im;        /* input image structure  */
VisXimage_t xim;       /* output image structure */
int        i,j;        /* index counters         */

    Vparse(&argc, &argv, par);     /* parse the command line       */
    VXin  = VXopen(IVAL, 0);       /* open input file              */
    VXout = VXopen(OVAL, 1);       /* open the output file         */

    VXlist = VXread(VXin);         /* read file                    */
    if (VXNIL != (vptr = VXfind(VXlist, VX_PBYTE))){
      VXsetimage(&im, vptr, VXin); /*   initialize input structure */
      VXmakeimage(&xim, VX_PSHORT, im.bbx, im.chan);
                                   /*initialize output structure   */

      for (i = im.ylo; i <= im.yhi; i++)
        for (j = im.xlo; j <= im.xhi; j++)
             xim.s[i][j] = im.u[i][j];

      VXwrite(VXout, xim.list);    /* write data                   */
    }
    VXclose(VXin);                 /* close files                  */
    VXclose(VXout);
    exit(0);
}
```

Figure 2: Example VisionX Program that uses image program tools

```
    /*  Advanced vint program                      */
    /* initial code is same as that in Figure 2   */

VisXimage_t im;        /* input image structure  */
VisXimage_t xim;       /* output image structure */
int        i,j;        /* index counters         */
VisXelem_t *vptr;      /* data element pointer   */

    Vparse(&argc, &argv, par);     /* parse the command line       */
    VXin  = VXopen(IVAL, 0);       /* open input file              */
    VXout = VXopen(OVAL, 1);       /* open the output file         */

    while ((vptr = vxlist = VXreadframe(VXin)) != VXNIL){ /* read a frame   */
      VXfupdate(VXout, VXin);                    /* update global constants */
      while (VXNIL != (vptr = VXfind(vptr, VX_PBYTE))){ /*  each byte image */
         VXsetimage(&im, vptr, VXin);          /*initialize input structure */
         VXmakeimage(&xim, VX_PSHORT, im.bbx, im.chan); /* output structure */
         for (i = im.ylo; i <= im.yhi; i++)
            for (j = im.xlo; j <= im.xhi; j++)
                xim.s[i][j] = im.u[i][j];

         /* replace the old image data with the new image data */
         VXmovelem(vptr, xim.list->next);  /* just move data, bbx is same  */
         vptr = VXdelelem(vptr);
         VXresetimage(&im);             /* deallocate the image structures */
         VXresetimage(&xim);
      }
      VXwriteframe(VXout,vxlist);              /* write frame             */
      VXdellist(vxlist);                       /* delete the frame        */
    }
    VXclose(VXin);                 /* close files                  */
    VXclose(VXout);
    exit(0);
}
```

Figure 3: Example program segment suitable for complex files

A Example VXparse Program for command line parameters

The following source code is a program that illustrates the use of `VXparse` to parse command line parameters.

```
/*************************************************************/
/* Example VisionX program that demonstrates the use of VXparse */
/*      and related tools for command line argument parsing     */
/* Syntax:                                                     */
/*        vpex if=infile of=outfile                            */
/*************************************************************/
#include "VisXV4.h"
#include "Vutil.h"
            /* Every VisionX program should have       */
            /* a "par" declaration that defines the    */
            /* prefixes for the command line arguments */
            /* vparse will fill the second element of  */
            /* the structure with the address of       */
            /* any matched prefixes in the command     */

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();

main(argc, argv)
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 = 0;    /* default for x */
    y = 0;    /* default for 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 -
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
```

B VisionX File and List Access Functions

Note that programs making use of these functions must include :

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

Also note that the following variables are assumed for the descriptions below :

```
VisXfile_t *a, *b;
VisXelem_t *e, *f;
VisXimage_t *i, *j;
int      s;
char    *c;
```

A summary of the VisX V4 file and list functions follows.

VisX File I/O Operations

```
    VisXfile_t * VXopen(c, s)       open a VisX File
    void         VXclose(a)         close a VisX file

  Whole File I/O
    VisXelem_t * VXread(a)          read a VisX File
    void         VXwrite(a, e)      write a VisX File

  Frame I/O
    void         VXfupdate(a, b)    update global constants
    VisXelem_t * VXreadframe(a)     read a VisX Frame
    void         VXwriteframe(a, e) write a VisX Frame
```

VisX Data List Manipulation

```
  List Functions
    VisXelem_t* VXinit()        Initialize an empty list
    void        VXdellist(a)    Delete a list

  Element Functions
    int         VXtype(a)       Return the type of an element
    int         VXbase(a)       Return the base type of an element
    int         VXsize(a)       Return the size of an element (bytes)
    void*       VXdata(a)       Return a pointer to the element's data
    int         VXnumelem(a)    Return the number of data items in an element

    VisXelem_t* VXfirst(a)      Return the first element of a list
    VisXelem_t* VXlast(a)       Return the last element of a list
    VisXelem_t* VXaddelem(a,<type>,<buffer>,<size>)
                                Add a new element to a list
    VisXelem_t* VXlnkelem(a,<type>,<buffer>,<size>)
                                Link a new element to a list
    VisXelem_t* VXdelelem(a)    Delete an element of a list
    VisXelem_t* VXfind(a, s)    Find an element of type s
    VisXelem_t* VXfindin(a, l)  Find an element of one of types l where
                                l is an array of types terminated by VX_NULLT
    VisXelem_t* VXbfind(a, s)   Find an element of type s searching backwards
```

```
void        VXjoin(a,b)     Joint list b to the end of list a
VisXelem_t* VXcpy(a)        Return a copy of a list

VisXelem_t* VXdelobject(a)  Delete a whole object from a list
```

Image Array Operations

```
int      VXsetimage(&i, a, f)  Create an Image structure from a file
int      VXmakeimage(&i, <type> ,<bounding box>, <channels>)
                               Create a new image structure
int      VXembedimage(&i, &j ,x1,x2,y1,y2) Create image with borders
int      VXfloatimage(&i, &j ,x1,x2,y1,y2) Create float image with borders
int      VXresetimage(&i )   Free memory allocated to an image structure
```

C VisionX Element Types

Each VisX type is identified by a unique numeric code and is given both a programing name and a text description. The following types are currently defined for VisX lists:

```
  Function       Code  Program Name

   Image Frames

Image Frame      0x1a  VX_FRAME
End Frame        0x2a  VX_EFRAME

Compact List     0x08  VX_COMP
2-D vector       0x33  VX_V2D
3-D vector       0x43  VX_V3D
Inverted Y       0x39  VX_INVY
Bounding Box     0x53  VX_BBX

  Color Vector r,g,b (float)
Color Triple     0x63  VX_COLOR

  Color Map r,g,b (byte)
Color Map        0x61  VX_CMAP
Object ID.       0x5a  VX_ID
Color Index      0x67  VX_CIDX
Feature Vec.     0x73  VX_GFV

  File Elements

File History     0x20  VX_FHIST
File Name        0x30  VX_FNAME
File Machine     0x50  VX_FMACH
File Command     0x40  VX_FCMND
File Date        0x4a  VX_FTIME
File User ID     0x3a  VX_FUID
```

```
  VisX Pixel types

Pixels (byte)    0xa1  VX_PBYTE
Pixels (short)   0xa2  VX_PSHORT
Pixels (float)   0xa3  VX_PFLOAT
Pixels (Double)  0xa4  VX_PDOUBLE
Pixels (S-byte)  0xa5  VX_PCHAR
Pixels (Bit)     0xa6  VX_PBIT
Pixels (Int)     0xa7  VX_PINT
Pixels (Index)   0xb1  VX_PIDX
Pixel Channels   0xaa  VX_PCHAN

  VisX Generic Measures

Generic(text)    0xc1  VX_GTEXT
Generic(byte)    0xc1  VX_GBYTE
Generic(short)   0xc2  VX_GSHORT
Generic(float)   0xc3  VX_GFLOAT
Generic(Double)  0xc4  VX_GDOUBLE
Generic(S-byte)  0xc5  VX_GCHAR
Generic(Bit)     0xc6  VX_GBIT
Generic(Int)     0xc7  VX_GINT
Generic(Scalar)  0xca  VX_GSINT


Not Defined      0x00  VX_NULLT
```