in /usr/include/argp.h

```
/* A description of a particular option.  A pointer to an array of
   these is passed in the OPTIONS field of an argp structure.  Each option
   entry can correspond to one long option and/or one short option; more
   names for the same option can be added by following an entry in an option
   array with options having the OPTION_ALIAS flag set.  */
struct argp_option
{
  /* The long option name.  For more than one name for the same option, you
     can use following options with the OPTION_ALIAS flag set.  */
  __const char *name;

  /* What key is returned for this option.  If > 0 and printable, then it's
     also accepted as a short option.  */
  int key;

  /* If non-NULL, this is the name of the argument associated with this
     option, which is required unless the OPTION_ARG_OPTIONAL flag is set. */
  __const char *arg;

  /* OPTION_ flags.  */
  int flags;

  /* The doc string for this option.  If both NAME and KEY are 0, This string
     will be printed outdented from the normal option column, making it
     useful as a group header (it will be the first thing printed in its
     group); in this usage, it's conventional to end the string with a `:'.  */
  __const char *doc;

  /* The group this option is in.  In a long help message, options are sorted
     alphabetically within each group, and the groups presented in the order
     0, 1, 2, ..., n, -m, ..., -2, -1.  Every entry in an options array with
```

```
32.         if this field 0 will inherit the group number of the previous entry, or
33.         zero if it's the first one, unless its a group header (NAME and KEY both
34.         0), in which case, the previous entry + 1 is the default.  Automagic
35.         options such as --help are put into group -1.  */
36.       int group;
37.     };
```

in driver/pip/pip-app/file_pip_adaptor.c

```
1.  static struct argp_option options[] = {
2.      { "object",         'b', "<DIR>",    0, "Directory that contains object tag s
    trips.  Code assumes object tag strips match in dimensions and number to contone
    strips." },
3.      { "hscale",         'h', "INT",      0, "Override to SharpM horizontal scale
    factor in 1000 units.  ie: 2500 will result in a 2.5x scale factor" },
4.      { "vscale",         'v', "INT",      0, "Override to SharpM vertical scale fa
    ctor in 1000 units.  ie: 2500 will result in a 2.5x scale factor" },
5.      { "discard",        'd', "0/1",OPTION_ARG_OPTIONAL, "Discard output data inst
    ead of writing to a file" },
6.      { "adf",            'a', "0/1",OPTION_ARG_OPTIONAL, "Pass in a height of 0 -
    simulates ADF behavior" },
7.      { "out_strip_height",'o',"INT",      0, "Height of the output strips.  Defaul
    t is 64." },
8.      { "output_directory",'D',"<DIR>",    0, "Directory to save the output.  Defau
    lt directory is /data/dump/" },
9.      { "skew",           's', "0/1",OPTION_ARG_OPTIONAL | OPTION_HIDDEN, "Check if
     output image is skewed" },
10.     { "pad_left",       'l', "INT",      0, "Number of pixels to pad to the left
    side of the image.  Padding occurs at end of pipe after clipping/scaling/halftoni
    ng." },
11.     { "pad_right",      'r', "INT",      0, "Number of pixels to pad to the right
     side of the image.  Padding occurs at end of pipe after clipping/scaling/halfton
    ing." },
12.     { "clip_left",      'c', "INT",      0, "Number of pixels to clip from the le
    ft side of the image.  Clipping occurs at start of pipe before scaling/halftoning
    /padding." },
13.     { "clip_right",     't', "INT",      0, "Number of pixels to clip from the ri
    ght side of the image.  Clipping occurs at start of pipe before scaling/halftonin
    g/padding." },
14.     { "cancel_strips",  'x', "INT",      0, "Cancel processing after <INT> number
     of input strips have been processed." },
15.     { "early_exit_pass",'X', "INT",      5, "Multi-pass early exit with strip wri
    te[1-5] 0 for no output." },
16.     { "calculate_crc",  'y', "0/1",OPTION_ARG_OPTIONAL,"Calculate CRC" },
17.     { "compare_crc",    'z', "CRC in HEX", 0,"Calculate CRC and compare to the CR
    C provided.  Program will exit with a value of 119 if CRC does not match." },
18.     { 0 }
19. };
```

in /usr/include/argp.h

```
/* An argp structure contains a set of options declarations, a function to
   deal with parsing one, documentation string, a possible vector of child
   argp's, and perhaps a function to filter help output.  When actually
   parsing options, getopt is called with the union of all the argp
   structures chained together through their CHILD pointers, with conflicts
   being resolved in favor of the first occurrence in the chain.  */
struct argp
{
  /* An array of argp_option structures, terminated by an entry with both
     NAME and KEY having a value of 0.  */
  __const struct argp_option *options;

  /* What to do with an option from this structure.  KEY is the key
     associated with the option, and ARG is any associated argument (NULL if
     none was supplied).  If KEY isn't understood, ARGP_ERR_UNKNOWN should be
     returned.  If a non-zero, non-ARGP_ERR_UNKNOWN value is returned, then
     parsing is stopped immediately, and that value is returned from
     argp_parse().  For special (non-user-supplied) values of KEY, see the
     ARGP_KEY_ definitions below.  */
  argp_parser_t parser;

  /* A string describing what other arguments are wanted by this program.  It
     is only used by argp_usage to print the `Usage:' message.  If it
     contains newlines, the strings separated by them are considered
     alternative usage patterns, and printed on separate lines (lines after
     the first are prefix by `  or: ' instead of `Usage:').  */
  __const char *args_doc;

  /* If non-NULL, a string containing extra text to be printed before and
     after the options in a long help message (separated by a vertical tab
     `\v' character).  */
```

```
32.     __const char *doc;
33.
34.     /* A vector of argp_children structures, terminated by a member with a 0
35.        argp field, pointing to child argps should be parsed with this one.  Any
36.        conflicts are resolved in favor of this argp, or early argps in the
37.        CHILDREN list.  This field is useful if you use libraries that supply
38.        their own argp structure, which you want to use in conjunction with your
39.        own.  */
40.     __const struct argp_child *children;
41.
42.     /* If non-zero, this should be a function to filter the output of help
43.        messages.  KEY is either a key from an option, in which case TEXT is
44.        that option's help text, or a special key from the ARGP_KEY_HELP_
45.        defines, below, describing which other help text TEXT is.  The function
46.        should return either TEXT, if it should be used as-is, a replacement
47.        string, which should be malloced, and will be freed by argp, or NULL,
48.        meaning `print nothing'.  The value for TEXT is *after* any translation
49.        has been done, so if any of the replacement text also needs translation,
50.        that should be done by the filter function.  INPUT is either the input
51.        supplied to argp_parse, or NULL, if argp_help was called directly.  */
52.     char *(*help_filter) (int __key, __const char *__text, void *__input);
53.
54.     /* If non-zero the strings used in the argp library are translated using
55.        the domain described by this string.  Otherwise the currently installed
56.        default domain is used.  */
57.     const char *argp_domain;
58.   };
```

```
/* The type of a pointer to an argp parsing function.  */
typedef error_t (*argp_parser_t) (int __key, char *__arg,
                                  struct argp_state *__state);
```

in driver/pip/pip-app/file_pip_adaptor.c

```
static struct argp argp= { options, parse_opt, args_doc, doc };
```

```
static struct arguments args;
```

```
struct arguments {
    int         mode;
    char        *dir_name;
    char        *object_tag_dir_name;
    uint32_t    hscale_override;
    uint32_t    vscale_override;
    bool        discard;
    bool        adf_mode;
    uint32_t    out_strip_height;
    bool        skew;
    uint32_t    pad_left;
    uint32_t    pad_right;
    uint32_t    clip_left;
    uint32_t    clip_right;
    uint32_t    calculate_crc;
    uint32_t    compare_crc;
    uint32_t    cancel_after_strips;
    uint32_t    early_exit_pass;
};
```

```
argp_parse( &argp, argc, argv, 0, 0, &args );
```

argp_parse()的目的是把输入的command line parameters的信息(argc, argv)
通过argp中的设置，parse成args中的可利用的有用信息。

```
/* Parse the options strings in ARGC & ARGV according to the options in ARGP.
   FLAGS is one of the ARGP_ flags above.  If ARG_INDEX is non-NULL, the
   index in ARGV of the first unparsed option is returned in it.  If an
   unknown option is present, ARGP_ERR_UNKNOWN is returned; if some parser
   routine returned a non-zero value, it is returned; otherwise 0 is
   returned.  This function may also call exit unless the ARGP_NO_HELP flag
   is set.  INPUT is a pointer to a value to be passed in to the parser.  */
extern error_t argp_parse (__const struct argp *__restrict __argp,
                           int __argc, char **__restrict __argv,
                           unsigned __flags, int *__restrict __arg_index,
                           void *__restrict __input);
```

这里没有使用
unsigned __flags,
int *__restrict __arg_index
这两个参数。

而最后一个输入参数__input则是用户定义的用于存放被解析后的信息的variable，这里就是struct arguments args。

argp_parse()会根据argp中的定义parse command line，具体怎么把parse出的parameter转变成output parameter args中的设置是由argp.parser callback function完成的。

```
static char args_doc[] = "<mode> <input_directory>";
```

```
static char doc[] = "Sample program to test the PIP block\n"
                    "Example command line: ./test_file_pip_adaptor 112 /data/tiger/\n"
                    "By default the generated output strips will be saved to /data/dump/\n"
                    "Exit Code status:\n"
                    "\t0 = success\n"
                    "\t119 = CRC comparison failed\n"
                    "\t120 = Input directory not found\n"
                    "\t121 = Failed to create output directory\n"
                    "\t122 = Object tag dir not found\n"
                    "\t123 = Object tag mismatch - # files or dimensions don't match contone data\n";
```

```
1.  static error_t parse_opt (int key, char *arg, struct argp_state *state )
2.  {
3.      struct arguments *arguments = state->input;        ①
4.      switch ( key ) {            ②
5.      case 'a':
6.          arguments->adf_mode = arg ? atoi( arg ) : 1;
7.          break;
8.      case 'b':
9.          arguments->object_tag_dir_name = arg;
10.         break;
11.     case 'c':
12.         arguments->clip_left = atoi( arg );
13.         break;
14.     case 'd':
15.         arguments->discard = arg ? atoi( arg ) : 1;
16.         break;
17.     case 'D':
18.         dump_path = arg;
19.         break;
20.     case 'h':
21.         arguments->hscale_override = atoi( arg );
22.         break;
23.     case 'l':
24.         arguments->pad_left = atoi( arg );
25.         break;
26.     case 'o':
27.         arguments->out_strip_height = atoi( arg );
28.         break;
29.     case 'r':
30.         arguments->pad_right = atoi( arg );
31.         break;
```

```
32.         case 's':
33.             arguments->skew = arg ? atoi( arg ) : 1;
34.             break;
35.         case 't':
36.             arguments->clip_right = atoi( arg );
37.             break;
38.         case 'v':
39.             arguments->vscale_override = atoi( arg );
40.             break;
41.         case 'x':
42.             arguments->cancel_after_strips = atoi( arg );
43.             break;
44.         case 'X':
45.             arguments->early_exit_pass = atoi( arg );
46.             break;
47.         case 'y':
48.             arguments->calculate_crc = arg ? atoi( arg ) : 1;
49.     break;
50.         case 'z':
51.             arguments->compare_crc = strtoul( arg, NULL, 16 );
52.             break;
53.         case ARGP_KEY_ARG:
54.             switch ( state->arg_num ) {
55.             case 0:
56.                 arguments->mode = atoi( arg );
57.                 break;
58.             case 1:
59.                 arguments->dir_name = arg;
60.                 break;
61.             default:
62.                 DBG_PRINTF_ERR( "Too many arguments\n" );
63.                 argp_usage( state );
```

```
            }
            break;
        case ARGP_KEY_END:
            if ( state->arg_num < 2 ) {
                DBG_PRINTF_ERR( "Too few arguments\n" );
                argp_usage( state );
            }
            break;
        default:
            return ARGP_ERR_UNKNOWN;
        }
        return 0;
    }
```

A pointer to a function that defines actions for this parser; it is called for each option parsed, and at other well-defined points in the parsing process. A value of zero is the same as a pointer to a function that always returns ARGP_ERR_UNKNOWN.

static struct arguments args;

中的args怎么会state->input的?

①

struct arguments *arguments = state->input;

这里state->input应该指向&args

②

这里的key应该就是struct argp_option options中的.key field。

而参数arg应该就是该key对应的value了。

比如command -b /temp/obj -a 1 -s 0

argp_parse()依据struct argp_option options中的信息，把上面的参数parse成了

key = 'b', value = "/temp/obj"

key = 'a', value = "1"

key = 's', value = "0"

argp.parser callback function,也就是这里的parse_opt()的责任就是解析这些key-value pairs。

具体parse字符串的责任被argp_parse() function完成了。