NAME

process_options – customizing options

SYNOPSIS

```
#include <ShoreApp.h>

main( int argc, char **argv,...)
{
   ...

   //
   // minimal required options handling ...
   // but don't do this if your program is called a.out
   // or anything with a "." in its name.
   //
   //
   // The SH_DO macro (defined in Shore.h) checks return codes
   //
   SH_DO(Shore::init(argc, argv));

   // Results in options being processed thus:
   //
   // Shore::default_options(argc, argv)
   // FILE: looks for .shoreconfig in . and then in ~
   // OPTIONS: names are shore.client.<argv[0]>.<option-name>

   // ...or...

   SH_DO(Shore::init(argc, argv, "myprog"));
   // overrides argv[0] with myprog so that
   // option names are shore.client.myprog.<option-name>
   //

   // ...or...

   SH_DO(Shore::init(argc, argv, 0, ".myrcfile"));
   // overrides .shoreconfig with .myrcfile
   // so that it looks for .myrcfile in . and then in ~

   // rest of main()
   ...
}

//
// OR...
// less simple but a little bit more flexible...
//

shrc    my_options(option_group_t *o);

main( int argc, ...)
{
   ...
```

```
    // this allows the application to add its
    // own options, to specify options class,
    // program name, name of file of commands, etc.

    char    *usage_string = "usage: how-to-use-this-command";
    shrc    rc;    // return code
    option_group_t *options;

    rc = Shore::process_options( argc, argv,
       // type is always "shore"
       "myclass",    // class of options
       "myprogram",// program name
             // - does not have to be argv[0]
             // options recognized will be those
             // of the form
             // shore.myclass.myprogram.<option-name>
             //
       ".rcfile",    // file to search-- can be null
             // search for ./.rcfile, then ~/.rcfile

       usage_string,     // string to print if error
       my_options, // optional - can be 0
       &options,   // return value returned here
       false         // I want to process arguments -h, -v
                 // here, using getopt()
       );

    if(rc){
        cerr << "Cannot process options:" << rc << endl;
        exit (1);
    }
    // ... my own processing of the rest of the argv,
    // e.g., using getopt()

    SH_DO(Shore::init()); // short form of init, since
                    // options were already processed

    // rest of main()
    ...
}
```

DESCRIPTION

When you are writing an application, you might wish to use the Shore options service, or you might handle your arguments and options your own way. Either way, your application program must provide the lower layers of Shore with the means to discover what values are to be used for the options that the Shore software uses. The simplest way to do this is to call the first form of the function **process_options,** which calls the lower layers of Shore to establish default values for all the options that have defaults, and looks (in a file and on the command line) for values for those options. If the given file name is a relative path name, the function **process_options** looks in the current directory and if no such file exists there, in the home directory (determined by `$HOME`). After processing the file, it reads the command line as determined by the first two arguments. A detailed description of these methods is in **init(oc).**

NB: Your application must be called by a name that does not include a period if you wish to set any options with a file with the 4-part command names. For example, `a.out` does not work with the 4-part command names but it does work with the wild-card command:

```
*.option: value
```

and `a.out` is fine if you do not need to set any options because the default values are satisfactory.

If you want your application to add its own options, to specify the program class or program name for options, or to read a configuration file other than that determined by **Shore::default_options** (see **init(oc)** for details), or if you do not want the command-line arguments *-h* and *-v* to be processed by the lower layers of Shore, you can have your `main()` call the function **process_options**. This calls the lower layers of Shore to establish default values for all the options that have defaults, and allows your program to create its own options. This takes place before the file and the command line are searched.

Each option has a name; its name is composed of four parts:

```
type.class.program.option
```

where *type* is `shore` , *class* and *program* are determined by the caller of **process_options**, and *option* is determined by the software layer that creates the option.

When a file or command line is searched for option values, the options are identified by their full 4-part names or by pattern-matching expressions that include wild cards ('*' and '?').

For example, to set the option `shore.oo7.bench.configfile` to the value `./script` in the program `bench`, the function `main()` in `bench` calls

```
...
Shore::process_options(argc, argv, "oo7", "bench",
    ".oo7rc", usage_string, oo7_options, &options, true)
```

and the file `.oo7rc` can contain any of these lines:

```
shore.oo7.bench.configfile: ./script
?.oo7.bench.configfile: ./script
shore.oo7.?.configfile: ./script
*.configfile: ./script
```

The `bench` program also has to add the option `configfile` to the list of options used by the lower layers of Shore. This is done by providing a non-null seventh argument to **process_options. Bench does this with its function oo7_options,** which **looks** like **this:**

```
shrc oo7_options(option_group_t *options)
{
    static option_t    *configfile_opt;
    shrc    e =
    options->add_option(
        "configfile",  // 4th part of option name
        "string",      // description of syntax of value
        "-",           // default value
        "name of configuration file", // meaning of option
        false,         // Boolean: is it a mandatory option?
        option_t::set_value_charstr,
                       // function to check the syntax of the value
        configfile_opt // return value - ptr to an option_t
                       // that describes this option
    );
```

```
    return e;
}
```

Finally, if you want even more control of the options processing, or if you want to take option values from the command line, you can use any of the functions from the options service, found in `#include <option.h>`.

BUGS

The options service is not completely documented. For more details, you'll have to look at the source code for something that uses options, like the Shore Value-Added Server or the oo7 example. The file that sets up the options is `svas_layer.C` and there is such a file in `src/vas/client` and another in `src/vas/server`. Search for `add_option`.

VERSION

This manual page applies to Version 1.1 of the Shore software.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518.

SEE ALSO

init(oc), options(oc), and **options(svas)**