NAME

      Options – Shore options - processing package

SYNOPSIS

```
#include <option.h>

class option_group_t {

            option_group_t(int max_class_levels);
            ˜option_group_t();

    w_rc_t   add_class_level(const char* name);

    w_rc_t   add_option(const char* name, const char* possible_values,
                        const char* default_value,
                        const char* description, bool required,
                        option_t::OptionSetFunc set_func,
                        option_t*& new_opt);

    w_rc_t   lookup(const char* name, bool exact, option_t*&);
    w_rc_t   lookup_by_class(const char* opt_class_name, option_t*&,
                        bool exact=false);

    w_rc_t   set_value(const char* name, bool exact,
                        const char* value, bool overRide,
                        ostream* err_stream);

    void     print_usage(bool longForm, ostream& err_stream);
    void     print_values(bool longForm, ostream& err_stream);
    w_rc_t   check_required(ostream* err_stream);
    w_rc_t   parse_command_line(char** argv, int& argc,
                        int min_len, ostream* err_stream);
    w_list_t<option_t>& option_list();
    int      num_class_levels();
    const char* class_name();
};

class option_t {
    bool        match(const char* matchName, bool exact=false);
    w_rc_t      set_value(const char* value, bool overRide,
                            ostream* err_stream);
    const char* value();
    bool        is_set();
    bool        is_required();
    const char* name();
    const char* possible_values();
    const char* default_value();
    const char* description();

    typedef w_rc_t (*OptionSetFunc)(option_t*, const char * value,
            ostream* err_stream);

    // Standard functions for basic types
    static w_rc_t set_value_bool(option_t* opt, const char* value,
```

```
                    ostream* err_stream);
        static w_rc_t set_value_long(option_t* opt, const char* value,
                    ostream* err_stream);
        static w_rc_t set_value_charstr(option_t* opt, const char* value,
                    ostream* err_stream);
        static bool str_to_bool(const char* str, bool& bad_str);
    };

    class option_file_scan_t {
                option_file_scan_t(const char* opt_file_path,
                                    option_group_t* opt_group);
                ~option_file_scan_t();
        w_rc_t  scan(bool over_ride, ostream& err_stream,
                                    bool exact=false);
    };
```

DESCRIPTION

A program uses the options package in these stages:

Establishes
    descriptions of options, default values, etc.

Scans
    a file and/or the command line for character-string representations of values chosen by the user.

Determines
    if all required options have been given values.

Parses
    the character-string representations of values given, and converting them to binary values.

Whether you are writing a value-added server or a Shore application, your program combines libraries that implement several software layers (or modules), each of which has its own set of options. It is the job of the function **main** to initiate each of the above steps, so that each software layer can perform the first step, then the file or command line can be scanned once to determine the values for all the layers' options. The options package determines if all required options have been given values, based on the options' descriptions created in the first step. Finally, each software layer performs the fourth step.

The first three steps are performed in proper succession by the function **process_options(oc), which is** in the client-side language-independent library. If are writing a value-added server, you can look at or use the function **::process_options** in the Shore Value-Added Server, found in in source tree at **src/vas/common/process_options.C.** If you want to write your own options-handling function, read on.

ESTABLISHING OPTION DESCRIPTIONS

An instance of *option_t* describes an option. It contains

name   a character string, the name of the option.

description a character string, describes the semantics of the option. Can be printed for "usage" and "help".

required   True if the option has no default value and the software that uses the option needs a value for the option.

set   True if a value has been given to this option (by default or otherwise).

value   Holds the last value given to the option, in the form of a character string (as typed on a command line or read from a file).

Options are grouped into option groups, represented by instances of *option_group_t*. By convention, each process has an option group, and each software layer or module adds options to the the option group.

An option group has a classification hierarchy associated with it. Each level of the hierarchy is given a string name. Levels are added with add_class_level(). The level hierarchy is printed in the form: 'level1.level2.level3.' A complete option name is specified by 'level1.level2.level3.optionName:'. A convention for level names is: *programtype.programname* where *programtype* indicates the general type of the program and *programname* is the Unix file name of the program.

Options are created and added to the group with the method **add_option,** and located in a group with the methods **lookup** and **lookup_by_class.**

```
option_t            *opt_make_tcl_shell;
option_t            *opt_nfsd_port;

option_group_t     options = new option_group_t(3); // 3 levels

W_DO(options->add_option("svas_tclshell",  // name of option
            "yes/no",  // help-information
            "yes",          // default value
            "yes causes server to run a Tcl shell", // help info
        false,          // ok if not set by user
        option_t::set_value_bool,  // function called during
                                // scan of options file or of command-line
                                // to check the syntax of the given value
            opt_make_tcl_shell      // place to stash a pointer
                            // to this option
        ));

W_DO(options->add_option("svas_nfsd_port",  // name of option
            "1024 < integer < 65535",     // help-information
        "2999",      // default value
            "port for NFS service",     // help information
        false,            // ok if not set by user
        option_t::set_value_long, // interpret strings as an integer
            opt_nfsd_port // place to stash a pointer
                            // to this option
        ));
```

SCANNING a FILE and COMMAND LINE

Given a group of options, a process can read a file containing option names and values, and set the values of the options in the option group accordingly. This is done with the class *option_file_scan_t*.

```
option_file_scan_t oscan(".shoreconfig", options);
w_rc_t e = oscan.scan(true, cerr); // override any
                                            // options already set
```

Applications might need to set option values explicitly, in which case they can do so with **option_group_t::set_value** or any of the static members of *option_t:* **option_t::set_value_bool, option_t::set_value_long, option_t::set_value_charstr.** These methods check the syntax of the character-string representations of values, but they do not convert the strings to binary values (Boolean, integer, etc.).

DETERMINING IF REQUIRED OPTIONS HAVE VALUES

The function **check_required** runs through all options associated with the option group, and determines if there is a value (default or assigned explicitly) for each one that was described in **add_option** as required.

PARSING VALUES

The application program or the function **main** must call functions to convert the character strings to values. Typically this is done as follows:

```
... = strtol(opt_nfsd_port->value(),0,0);

// or

bool  isbad=false;
... = option_t::str_to_bool(opt_nfsd_port->value(),isbad);
if(isbad) {
        // if this returns isbad==true, set_value_bool()
        // would have returned an error had it been called previously
}
```

ERRORS

Errors returned from the option method are:

```
OPT_IllegalDescLine     - Illegal option description line
OPT_IllegalClass        - Illegal option class name
OPT_ClassTooLong        - Option class name too long
OPT_TooManyClasses      - Too many option class levels
OPT_Duplicate           - Option name is not unique
OPT_NoOptionMatch       - Unknown option name
OPT_NoClassMatch        - Unknown option class name
OPT_Syntax              - Bad syntax in configuration file
OPT_BadValue            - Bad option value
OPT_NotSet              - A required option was not set
```

VERSION

This manual page applies to Version 2.0 of the Shore Storage Manager.

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. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

COPYRIGHT

SEE ALSO

process_options(oc)