

## 4.0 FEMIS Command Server

Command server online documentation is provided in three man pages on the UNIX server. Log onto the EOC's server as `femis` and enter:

```
% man cmdservd
% man cmdserv.conf
% man cmdserv
```

`cmdservd` is the command server daemon. `cmdserv.conf` is the command server configuration file. `cmdserv` is a UNIX test client for the command server.

### 4.1 cmdservd – FEMIS Command Server Daemon

#### 4.1.1 Synopsis

```
cmdservd [-conf config-file]
cmdservd [-conf config-file] [-v] [-syntax [-show] [-check]]
```

#### 4.1.2 Availability

The FEMIS command server daemon `cmdservd` executable, configuration file, test client, and related files are included with the FEMIS application. The default locations for these files are `/home/femis/bin` and `/home/femis/etc` on the FEMIS UNIX data server.

#### 4.1.3 Description

FEMIS utilizes remote command servers, executing on a UNIX host computer so PC workstation users can launch large mathematical model/simulation programs. These include the Evacuation SIMulation (ESIM), a module in the Oak Ridge Evacuation Modeling System (OREMS).

The command server is also utilized in certain FEMIS system administration functions, e.g., starting-stopping notification.

A high degree of security is realized in this command server because:

- Security problematic command servers such as `rsh` and `.rhosts` are not used. A client node need not be a trusted host.
- A command server runs only as a non-privileged, non-root process.
- A command server is forked as a child of `inetd`, eliminating the need to maintain socket connections.

- The command server does not execute raw UNIX commands. Rather it looks up necessary commands in a configuration file and matches parameters with arguments based on messages from the client.
- The command server is very limited in what it can do. Only those commands and functions defined in the `cmdservd.conf` configuration file can be invoked.
- Files written are only those temporary and output files written by the target executable. All communication between command server and forked process takes place via memory and unnamed pipes only.
- Passwords and other sensitive data are sent on the client-to-server socket encrypted. Clear passwords are never sent to the application on the command line to possibly be displayed by `ps`.
- The user and client machine making requests to run programs on a command server are verified prior to running any entry. Several methods are utilized to block requests from anyone except authorized users.

#### 4.1.4 Options

The command server has two basic execution modes: daemon and command line. In daemon mode, execution is started and controlled by the *inetd* Internet daemon and runs as a detached process. In command line or interactive mode, `cmdservd` runs in response to a user entry. Command line mode is used mainly to check on the syntax of new configuration files.

The default configuration file name is `cmdservd.conf`, and its default path is `/home/femis/etc`. To change either the configuration file name or path, use the `-conf` option. Possible formats for use with the `-conf` option are as follows:

```
1% cmdservd -conf filename
2% cmdservd -conf subdirectory/
3% cmdservd -conf subdirectory/filename
4% cmdservd -conf /fullpathname/
5% cmdservd -conf /fullpathname/filename
```

Case 1 Syntax contains no slashes ( / ), and thus no path or directory names. The argument to `-conf` is the name of a file which resides in the default configuration directory `/home/femis/etc`.

Case 2 Syntax is in subdirectory format and contains a slash ( / ) as the last character. The first character is not a slash and comma ( /, ) thus a relative path and not an absolute path. The described syntax tells `cmdservd` to use the default file name in a subdirectory of the default path.

Case 3 Syntax specifies a subdirectory and file name. The named file is thus located in the subdirectory of the default path.

Case 4 Syntax specifies to look for the default file name `cmdservd.conf` in the full path specified in the option. Both first and last character of the option are slashes ( / ).

Case 5 Syntax specifies a full path and file name. None of the defaults apply in this case.

Option `-v` asks `cmdservd` to display its version information. Example:

```
virus% cmdservd -v
cmdservd version 1.0 - Wed Feb 14 14:41:00 PST 1996
```

Option `-syntax` invokes only the `cmdservd` syntax checker.

Options `-show` and `-check` are used in conjunction with `-syntax`.

The `-syntax -check` options cause `cmdservd` to process the configuration file, checking for syntax problems. Options `-syntax -show` cause `cmdservd` to compile the configuration file, check for syntax problems, and display the resulting linked structure.

## 4.1.5 Syntax Check

To check the syntax of a command server configuration file, enter the options `-syntax -check` to `cmdservd`, examples:

```
1% cmdservd -syntax -check          # check default
2% cmdservd -syntax -check -conf CFG # check CFG file
```

The following format is output by `-syntax -check`. Any line detected with suspect syntax is reported.

```
Line ##: line-from-file
        error-message
        error-message
```

where `##` is the line number, `line-from-file` is the text from the configuration file at line `##`, and `error-message` is a list of error messages describing the problems. Example:

```
Line 13: badnews
        invalid block/directive type code
```

The following list provides all possible error messages and their probable cause.

invalid block/directive type code

A block name or directive name is not one of those allowed. The block names are ALL, ACCESS, HOST, SITE, and ENTRY. Directive names are site, deny, allow, executable, directory, password, outfile, errfile, argument, environment, file, and put.

block requires no parameters

The ALL and ACCESS blocks do not require a list of parameters, i.e., [BLOCKNAME par1 par2 ...].

block requires exactly 1 parameter

The ENTRY block requires exactly one parameter which is the entry item name, e.g., [ENTRY abc], where abc is the name of a program.

block requires 1 or more parameters

The HOST and SITE blocks require at least one parameter which is a list of host or site names. HOST and SITE cause conditional compile. If the current host or site is the same as an item in the list, compilation continues. Otherwise, compilation of this program block is blocked.

directive not valid outside a block

All directives must be contained inside a block.

ENTRY block can not include other blocks

It is invalid for an [ENTRY ..] block to contain other blocks (at this time).

directive must be inside HOST block

The site directive is only valid inside a HOST block.

directive must be inside ACCESS or ENTRY block

The allow and deny directives are only valid inside for an ACCESS or ENTRY block.

directive must be inside ENTRY block

Directives executable, directory, password, outfile, errfile, file, put, and argument are only valid inside an ENTRY block.

environment must be inside ENTRY ALL SITE or HOST block

The environment directive must be inside of an ENTRY, ALL, SITE, or HOST block. When inside ENTRY, the variable is evaluated for that entry item only. When inside ALL, SITE, or HOST, the variable is evaluated whenever the block condition is TRUE, and not evaluated if the block condition is FALSE.

ACCESS block can only contain deny and allow

An ACCESS block can not contain anything but deny and allow.

site requires exactly 1 parameter

site directive requires exactly one parameter. Zero parameters and two or more parameters are invalid syntax.

directive requires 1 or 2 parameters

allow and deny directives require exactly one or two parameters. Zero parameters and three or more parameters are invalid syntax.

invalid character(s) in IP address field

Internet Protocol (IP) address field in the deny and allow directives can contain only digits 0-9 and the period ( . ) characters. Anything else is invalid syntax. A format specification is not valid in allow or deny directives.

invalid character(s) in IP subnet mask

IP subnet mask in a deny or allow directive can contain only digits 0-9 and the period ( . ) characters. Anything else is invalid syntax.

invalid IP address

IP address numbers must be in the range 0-255.

invalid IP subnet mask

Only the numbers 255, 254, 252, 248, 240, 224, 192, 120, and 0 are valid IP subnet mask elements. The value 0 must be followed by 0. The value 255 must be preceded by 255. A value not 0 or 255 can appear only once. For example, 255.255.255.192, 255.255.255.0, 255.255.128.0.

directive requires format [parameters]

Directives executable, directory, password, outfile, errfile, file, put, argument, and environment require a format string and an optional list of parameters. Examples:

```
executable /home/femis/bin/command/xyz
directory /home/femis/user/%s/ DIRECTORY
```

only %s allowed in format

Format strings in this language allow only the %s printf conversion. Conversions, such as %d, %x, and %u are not allowed.

format and number of parameters do not match

The number of parameters included and the number required by the format string do not agree.

executable path/file affected by client

Structure of the configuration file program that generates the executable path/file string is affected by external environment variables sent in the client message. Such affects are illegal. Executable must be developed only from static values and environment variables local to the configuration file.

password affected by client

Structure of the configuration file program that generates the password string is affected by external environment variables sent in the client message. Such affects are illegal. The password must be developed only from static values and environment variables local to the configuration file.

## 4.1.6 Installation

The installation process copies files `cmdservd`, `cmdserv`, and `cmdserv.conf` to directory `/home/femis/bin` and `home/femis/etc`. These files are required to be at this path, unless modifications are made to the `/etc/inetd.conf` and `cmdserv.conf` files.

FEMIS installation adds the following line to the `/etc/services` file to define the command server service port name.

```
femis-cmdserv 9015/tcp fxcmdserv # command server
```

FEMIS installation adds the following single line to the `/etc/inetd.conf` file to add the command server to the *inetd* Internet daemon.

```
fxcmdserv stream tcp \
    nowait femis /home/femis/bin/cmdservd cmdservd
```

## 4.1.7 Protocol

Only Transmission Control Protocol (TCP) connection and reliable messages are ever used in the FEMIS command server daemon (`femiscomd`). User Datagram Protocol (UDP) is not used.

The FEMIS command server and a client program carry on a two way half duplex conversation. After successful connection has completed, the server and client exchange hello messages. The server hello message contains encryption seeds for the session. The client hello message contains optional mode flags, used to characterize certain server-client exchanges.

Once hello messages have been exchanged, `cmdservd` then listens for command messages from the client which contain the necessary parameters and instructions for running a specific program on the UNIX server.

After receiving a command, the command server looks for that entry in the configuration file. Actual UNIX commands and the format of arguments come from the configuration file, not from the socket input.

After completing the set up for a computation, the `femiscomd` forks and executes the specified executable and then goes back to listening for client commands.

## 4.1.8 Messages

This section describes messages that pass between server and client over TCP socket connections.

### 4.1.8.1 Message Format

Messages to/from command server and its client have the following general format.

```
<op:OPERATION|...|...|...><NEWLINE>
```

Every message begins with `<` and ends with `>` followed by an end-of-line. Only characters between `<` and `>` have any meaning. The end-of-line character, and anything between `>` and `<` have no meaning and should be ignored by both client and server.

Between `<` and `>` are an unspecified number of fields, the first one being the operation field. Fields are separated by the pipe ( `|` ) character. Fields can contain any number of characters or may be empty, i.e., `||`.

Within a field, four characters are escaped: < > | and \. The back slash ( \ ) is the escape character.

**Note:** The field separators < > and | never appear in a correctly encoded field.

The following mappings apply.

Decoded	Encoded
<	\L
>	\R
	\D
\	\E

### 4.1.8.2 Message Fields

All message field identifiers are two lower case characters followed by a colon. The identifiers are as follows:

Field	Contents
op:	Operation or function name
ac:	Action code: run, status, kill
pw:	Password field
ev:	Parameter (environment) values
rc:	Return code
er:	Error code
k0:	Key #0 for light encryption (not used)
k1:	Key #1 for light encryption (not used)
k2:	Key #2 for light encryption (not used)
mo:	Modes: alert test ... (client hello only)

### 4.1.8.3 Operation Codes

The current message operation codes currently are implemented in the command server, the command server's test client, or both:

Code	Description
op:SVRHELLO	Server hello
op:CLIHELLO	Client hello
op:MISCINFO	Miscellaneous info
op:EOF	End-of-file
op:COMMAND	Command directive
op:HELP	Help
op:HELPIFNO	Help information
op:QUIT	Quit

op:ERROR	Error to client
op:REPLY	Reply to client
op:ALERT	Alert the client

#### 4.1.8.4 Command Message

```
<op:COMMAND|ac:ACTION|pw:PASSWD|ev:PAR1|ev:PAR2|...>
```

where ACTION is run ENTRY, status, or kill; PASSWD is a password string; PAR1 and PAR2 are parameter defines; and ENTRY is the name of an entry in the configuration file.

This message is constructed by the client and sent to the server. It tells the server what entry from the configuration file to invoke. It tells the server what values to use for arguments and environments.

The PASSWD password string should be blank if the entry contains no password definition. If password is present, it must be a 16+ characters password value. The first eight characters are the HWID hex value. The next eight characters are the client port hex value. Following characters are the user's password string.

Parameters are utilized in the command server as environment variables. Each parameter specification PAR1 PAR2 defines an environment variable, e.g., X=1, CRANK=24-99, NAME=xyz, DB=CTOO. The environment variables thus defined are passed to the configuration file processing and become inputs for building application arguments, input files, and environment. Also see *cmdserv.conf* man page.

#### 4.1.8.5 Error Messages

```
<op:ERROR|er:MESSAGE>
```

where MESSAGE is the error message from the command processor.

The following lists possible errors.

```
can't access client data
can't access client data: PERROR
- Call to getpeername(socket) failed.
- PERROR is message returned from perror().

config file open failed
config file open failed: PERROR
- Open the configuration file failed.
- PERROR is message returned from perror().

config file syntax error on lines LINELIST
- Execution of command server has been terminated because there is one or
  more syntax errors in the configuration file.
```

- LINELIST is a list of line numbers with errors.
- Correct the syntax errors and retry. Use -syntax and -check options to see details of the syntax problems.

access denied

- The configuration file allow and deny directives in ENTRY or ACCESS block on the server host ban this command (or all) from client's IP address.

invalid command

- Content of message is not a valid command.

no action

- No valid action was specified.

no password

- A password is required and none was sent.

wrong password prefix

- Either HWID or PORT has wrong value.

unknown action

- Action code in COMMAND message not valid.
- Valid actions are run status kill.

wrong password

- Password supplied is not one required by configuration file.

can't set directory

can't set directory: PERROR

- Cannot change directory to the one specified.
- PERROR is message returned by perror().

already active

- The command server daemon is already executing a process. Either kill or wait for alert.

can't execute program

- Either fork() or execvp() failed. This probably happened because there's something wrong with the executable file or the name specified.

no executable

- The named executable file was not found. There may be something wrong with the path, or the file name.

#### 4.1.8.6 Reply Messages

<op:REPLY|rc:MESSAGE>

where MESSAGE is the reply message from the command processor.

The following lists possible replies.

```
successful
- command completed successfully

finish TIMESTAMP IDENT
- STATUS is execution finished
- TIMESTAMP also used in log file names
- IDENT is the UNIX process id number

killed TIMESTAMP IDENT
- STATUS is execution killed
- TIMESTAMP also used in log file names
- IDENT is the UNIX process id number

active TIMESTAMP IDENT
- STATUS is execution still in progress
- TIMESTAMP also used in log file names
- IDENT is the UNIX process id number

not active
- No process has been executed.
```

#### 4.1.8.7 Alert Message

```
<op:ALERT|rc:MESSAGE>
```

where MESSAGE is the process completion status:

```
finish TIMESTAMP IDENT
- STATUS is execution finished
- TIMESTAMP also used in log file names
- IDENT is the UNIX process id number

killed TIMESTAMP IDENT
- STATUS is execution killed
- TIMESTAMP also used in log file names
- IDENT is the UNIX process id number
```

#### 4.1.8.8 Message Example

```
From server      From client
<op:MISCINFO|ITEM1|ITEM2|...>
<op:SVRHELLO|k0:|k1:|k2:>
      <op:CLIHELLO|k1:|k2:|mo:alert>
      <op:COMMAND|ac:run test|
      pw:|ev:A=73|ev:B=Dog|ev:X=Cat>
<op:REPLY|rc:active 9602141130 12933>
      <op:COMMAND|ac:status|pw:>
<op:REPLY|rc:active 9602141130 12933>
      <op:COMMAND|ac:status|pw:>
```

```
<op:REPLY|rc:active 9602141130 12933>  
<op:ALERT|rc:finish 9602141130 12933>
```

## 4.1.9 Service Port and Name

The `cmdservd` service port number currently is 9015. The short name is `femis-cmdserv` or `fxcmdserv`.

## 4.1.10 Files

Files utilized during the installation and execution of the FEMIS command server include the following:

- `/home/femis/bin/cmdservd` daemon executable
- `/home/femis/etc/cmdserv.conf` configuration file
- `/home/femis/bin/cmdserv` test client (UNIX)
- `/etc/services` service port numbers
- `/etc/inetd.conf` internet daemon config

## 4.2 `cmdserv.conf` – FEMIS Command Server Configuration File

### 4.2.1 Availability

The FEMIS command server configuration file `cmdserv.conf` is included with the FEMIS application. The default location of the file is `/home/femis/etc` on the FEMIS UNIX data server.

### 4.2.2 Description

This configuration file provides specific configuration information to the FEMIS command server daemon `cmdservd`. Unlike problematic remote compute servers such as RSH, the FEMIS command server provides some degree of security through this configuration file.

Security is also realized by placing severe limits on what this command server is allowed to do. Only those procedures defined in the configuration file can be spawned.

Additional security is realized through an encrypted password mechanism. `cmdservd` currently uses simple encryption, with RSA or SSL planned for the future.

The FEMIS project and a CSEPP site administrator have the ability to configure allowed and denied clients on a per site basis. Allow and deny directives give the administrator the ability to allow individual workstations in the local Emergency Operation Center (EOC), or a remote EOC, but deny all others. Specification of allowed and denied workstations is based on IP address.

The processes used in the command server daemon to parse its configuration file are similar to how LEX/YACC generated parsers work. In LEX, a parser reads text according user defined rules. Output of the LEX analyzer is handed to the compiler YACC that builds a complex linked structure. The linked structure provides a simple mechanism for the process to scan the input program without having to reread and reparse the input files.

In the command server daemon, the source code is read by a text parser function. This parser recognizes only two general source constructs: block and directive. Block is the outer level construct and directive the inner level. A block can contain other blocks or directives. Directives are stand-alone—they do not contain other directives or blocks.

### 4.2.3 Syntax

A configuration file contains block, directive, and comment syntax constructs.

A line starting with a # character in column 1 is a comment. Any # character, not part of a string, begins a comment to the end of that line. Example:

```
# a comment line
argument %s XYZ   # comment to end-line
argument %s YZX   # another comment ...
```

A block identification begins with the [ (left bracket) character and ends with ] (right bracket). All blocks are terminated by [END] . General block syntax is as follows:

```
[BLOCK]   or   [BLOCK parameters]
...
[END]     [END]
```

Directive lines begin with a keyword, followed by zero or more parameters. Directive parameters can be additional keywords, or a quoted string. General directive syntax is as follows:

```
directive
directive parameter
directive format-string
directive format-string parameters
```

General syntax of a command server configuration file is as follows:

```
# comments
[BLOCK declaration]
directives
more blocks
[END]
more blocks
```

## 4.2.4 Block Syntax

The command server configuration language utilizes five block types: ACCESS, ENTRY, HOST, SITE, and ALL. A block statement always begins with the [ (left bracket) character, is followed by the block type name, and ends with ] (right bracket). Whether parameters are required is a function of block type.

The block types and their summary purpose are as follows:

Block Type	Purpose
[ACCESS]	Begin access specification block
[ENTRY entname]	Begin entry block (conditional)
[HOST hostlist]	Begin host block (conditional on host)
[SITE sitelist]	Begin site block (conditional on site)
[ALL]	Begin unconditional block
[END ...]	Marks end of a block

In ACCESS block, a parameter after the block type is not required nor is one allowed. Likewise, the ALL block does not require a following parameter, nor is one allowed.

An ENTRY block requires one and only one parameter, the entry name.

The HOST and SITE blocks require a list of one or more parameters, where the parameters are names of hosts or names of sites.

The END statement must have the characters [ENDxxx], where xxx is zero or more unprocessed characters, i.e., the parser scans only for [END. Characters xxx are only for commentary purposes, i.e., [END of block]. Every block must be terminated by an [END] statement, which marks the end of the block.

A simple example of command server configuration file structure follows:

```
#
# a comment line
#
[HOST princess queen] # if host is princess or queen
[ENTRY travelcost]    # then define entry travelcost
...
[END of travelcost]
[ENTRY distance]      # and define entry distance
...
[END of distance]
[END of princess queen]
```

The following sections contain detailed descriptions of each block type.

#### 4.2.4.1 ACCESS Block

Through an `ACCESS` block, the FEMIS project or a CSEPP site administrator can configure allowed and denied access to command server resources on a site's UNIX data server.

Two (and only two) directives are permitted in an `ACCESS` block: `allow` and `deny`. The `ENTRY` block also permits `allow` and `deny` directives.

When `allow` and `deny` appear in an `ENTRY` block, they specify what workstations can execute the specific entry. When `allow` and `deny` appear in an `ACCESS` block, they specify what workstations can execute any entry in the configuration file. An `ACCESS` block may be placed inside of `HOST` or `SITE` blocks, thus adding site-by-site conditional use.

The parameters of `allow` and `deny` directives are in the form of an IP address. This parameter can be in the form of a specific host address or a subnet designation.

The parameters of `allow` and `deny` can be a full absolute IP address, a partial IP address with an assumed mask, or an IP address with a mask. The assumed mask is `255.255.0.0` or `255.255.255.0`. At this time, only subnet masks `255.255.0.0` and `255.255.255.0` have any meaning. A zero in any field of the IP address means wild card.

Correct use is to first deny everything via `deny 0.0.0.0` and then one at a time allow subnets and/or specific IP addresses that exist at the site or EOC.

An address match refers to the client computer's IP address. If the client IP address Boolean-anded with the IP mask equals the IP address in the `allow` or `deny` directive, the match is set `TRUE`. If they are not equal then `FALSE`.

The following example allows access by all IP addresses on the PNL-Net, except for workstations `wd_millard` and `merlin`. Access by addresses on the PNL-Remote subnet (remote dial-in) are also allowed. The entire world outside PNL-Net and PNL-Remote are denied access.

```
[SITE PNL]
[ACCESS]
deny 0.0.0.0      # deny world
allow 130.20.0.0 # allow pnl-net...
deny 130.20.92.40 # deny wd_millard
deny 130.20.76.40 # deny merlin
[END of ACCESS]
[END of PNL]
```

#### 4.2.4.2 HOST Block

The format of a `HOST` block declaration is

```
[HOST host1 host2 host3 ...]
```

where: `host1 host2` is a list of one or more host names.

The `HOST` block is a conditional block which is compiled only if the server host, on which the command server daemon `cmdservd` is executing, is contained in the list of permitted hosts, i.e., the `HOST` block parameter list.

The following example defines the site to be `PNNL`, only if the name of the command server host is `virus`, `locusts`, `temblor`, or `mirage`. The example code fragment also sets up access for the site.

```
[HOST virus locusts temblor mirage]
site PNNL # site name is PNNL
[END]
[SITE PNNL]
[ACCESS]
deny 0.0.0.0 # deny whole world
allow 130.20.92.0 # allow isb1-400-pod subnet
allow 130.20.194.0 # allow pnl-femis-1 subnet
allow 130.20.210.0 # allow pnl-femis-2 subnet
allow 130.20.226.0 # allow pnl-femis-3 subnet
allow 130.20.242.0 # allow pnl-femis-4 subnet
[END]
[END]
```

#### 4.2.4.3 SITE Block

The format of a `SITE` block declaration is

```
[SITE site1 site2 ...]
```

where: `site1 site2` is a list of one or more site names.

The `SITE` block is a conditional block that is compiled only if the server host, on which the command server daemon `cmdservd` is executing, is within one of the sites listed. The specific site is determined by the site directive.

In the following example, the `ENTRY` definitions are compiled only if the local host is in one of the named sites: `PNNL`, `TEAD`, and `UMDA`.

```
[SITE PNNL TEAD UMDA]
[ENTRY import]
...
[END]
[ENTRY execute]
...
[END]
[END]
```

#### 4.2.4.4 ALL Block

The command server configuration file syntax rules require that all directives be contained inside of a block. Thus, a directive cannot be placed at the outer most level, as only blocks are allowed at that level.

In most cases, directives are not needed except inside blocks. However, there are special cases where placing a directive at the outer most block is necessary. The `ALL` block effectively allows that case. The `ALL` block is like a conditional block that is always `TRUE`. It might be used where a `HOST` or `SITE` block would be used, however the `ALL` block always compiles.

One special case that requires an `ALL` block is definition of global environment variables. Consider the following example.

```
[ALL]
environment DATABASE fi7
[END]
[HOST virus]
environment DATABASE fi6
[END]
```

In the example above, environment database is first defined to be `fi7`, all the time. Then if the host is `virus`, `DATABASE` is redefined to be `fi6`.

#### 4.2.4.5 ENTRY Block

An `ENTRY` block defines a block of code that is used in the command server to set up the execution of a child subprocess. The command, script, or executable to be spawned can be a compiled program, a Bourne script, a C Shell script, or a PERL script.

The executable directive tells the command server where to find the entry's application file. Other directives set up arguments, parameters, and data being passed to the application.

The directive types permitted within an `ENTRY` block are as follows:

```
executable, directory, password, outfile, errfile, argument, environment, file,
put, allow, and deny.
```

The parameter in the `ENTRY` statement is the entry name, which the command server matches with the parameter in a run command message from a client. See *cmdserved(1)* man page. Example:

```
<op:COMMAND|ac:run entry-name|...>
```

## 4.2.5 Directive Syntax and Semantics

In the command server configuration language, blocks define the structure of a configuration program, and directives define actions to be executed at some point.

Directives are coded on a single line, which does not begin with the [ (left bracket) or # (comment) character. Generally, a directive consists of the directive type name, followed by an optional format statement, followed by one or more parameters.

Directives utilize a format string which appears much like the format strings of the C programming language. In this language, only the %s conversion type is valid, i.e., %d %x %u are not supported and, if included in a format, produce an error. Any number of %s conversions can appear in a format string. This is the way in which data from the client program is passed on to the application.

The parameters in a directive statement can be a simple string or the name of an environment variable. Environment names utilized get their values from the COMMAND:run messages from a client. In the example below, variables A, B, and C get values 1, 73, and 88X. All values are string values. Example:

```
<op:COMMAND|ac:run x|ev:A=1|B=73|C=88X|...>
```

Following is a table of directives in the command server language:

Directive	Purpose
site	Define the name of a site
executable	Define name of executable file
directory	Define default directory
password	Define password
outfile	Name the stdout file
errfile	Name the stderr file
argument	Specify a command line argument
environment	Specify an environment variable
file	Open and write a file
put	Put record into opened file
allow	Allow access by client
deny	Deny access by client

Three methods have been provided in the command server configuration language for copying input parameters to the application: argument, environment, and file/put. Argument generates an application command line argument. Environment creates an environment variable that then gets duplicated in the application. File/put create a file that can be read by the application.

### 4.2.5.1 Site Directive

The `site` directive defines the name of the site. This site name is then utilized in `SITE` blocks to conditionalize other blocks.

The `site` directive is only valid inside a `HOST` block. Example:

```
#
[HOST virus locusts temblor mirage]
site PNL
[END]
#
[HOST cemsun tcemsun]
site UTAH
[END]
#
[SITE PNL]
environment DATAPATH /files3/home/femis/data/pnl/
[END]
[SITE UTAH]
environment DATAPATH /files1/home/femis/data/utah/
[END]
#
[ENTRY xyz]
...
argument %s DATAPATH
[END]
```

**Note:** The same thing could be accomplished by using only the `HOST` block. However, `SITE` provides a convenient shorthand way to group a list of hosts that exist at the different CSEPP sites.

In the example above, the environment variable `DATAPATH` is changed depending on site value. Placing the definition of `DATAPATH` outside the `ENTRY` blocks helps to decrease the amount of configuration file code necessary.

### 4.2.5.2 Executable Directive

The `executable` directive provides the command server daemon with the executable file name. Possible formats are

```
executable file-name
executable format parameter-list
```

where `file-name` is an absolute. Only the string data type is supported—no integer or floating data.

`Format` is a `cmdserv` allowed format (see above). `Parameter list` is a list of internal environment variable names. The number of environments in the list must match the number of `%s` designators in the format string.

The `executable` directive requires that the environment variables used to generate the file name must be internal only. For this directive, external (client) environments are not allowed. The command server daemon does not allow the client to override the value of a previously specified environment if that environment is then used in the name of an executable, which would constitute a significant security hole. Examples:

```
executable /home/femis/bin/import.sh

environment EXEPATH /home/femis/bin/esim/
executable %s/import.sh EXEPATH
```

In the examples above, the first example is valid because it is static and does not involve environments. The second example also is valid, provided the client does not override the value of environment `EXEPATH`.

### 4.2.5.3 Directory Directive

The `directory` directive provides the command server daemon with the path to use for current directory prior to running the application. See *chdir(2)* man page. Possible formats are

```
directory path-name
directory format parameter-list
```

where `path-name` is an absolute. Only the string data type is supported—no integer or floating data.

`Format` is a `cmdserv` allowed format (see above). `Parameter-list` is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of `%s` designators in the format string.

If `cmdservd` can not set `directory` to the specified path, it returns an error message to the client, and does not run the application.

### 4.2.5.4 Password Directive

The `password` directive provides the command server daemon with the password to use for this application. The password string can be blank. If the `password` directive is omitted, it is assumed to be blank. A blank password means that password checking is not performed in `cmdservd` prior to running the application. Possible formats are

```
password password-string  
password format parameter-list
```

where `password-string` is the full password specification. Only the string data type is supported—no integer or floating data.

`Format` is a `cmdserv` allowed format (see above). `Parameter-list` is a list of internal environment variable names. The number of environments in the list must match the number of `%s` designators in the format string.

The `password` directive requires that the environment variables used to produce the password string must be internal only. For this directive, external (client) environments are not allowed. The command server daemon does not allow the client to override the value of a previously specified environment if that environment is then used in a `password` directive, which would constitute a significant security hole because the client could specify its own password.

If the `password` directive specifies a non-blank string, `cmdservd` then requires the client to send a password string in the `COMMAND` message. That password must match the one generated in the `password` directive. If a match is not realized, `cmdservd` returns an error message to the client, and does not run the application. Examples:

```
password georgewashington  
  
password Elisabeth-2  
  
environment SPORT Baseball  
environment TEAM SeattleMariners  
environment PLAYER Ichiro  
password %s-%s TEAM PLAYER
```

The first and second examples specify valid passwords because they are static and do not involve any environments. The third example also is valid, provided the client does not override the value of environments `TEAM` or `PLAYER`.

#### 4.2.5.5 Outfile Directive

The `outfile` directive tells the command server daemon the file name of where to write the application's standard output. If no `/path` is included in the `outfile` directive, the file will be written to the default directory.

If `outfile` and `errfile` specify the same string, only one file is created and `stdout` and `stderr` point to the same descriptor.

Possible formats are

```
outfile file-name  
outfile format parameter-list
```

where *file-name* is a full or partial file specification. Only the string data type is supported—no integer or floating data.

*Format* is a *cmdserv* allowed format (see above). *Parameter-list* is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of *%s* designators in the format string.

#### 4.2.5.6 Errfile Directive

The *errfile* directive tells the command server daemon the file name of where to write the application's standard error. If no */path* is included in the *errfile* directive, the file will be written to the default directory.

If *errfile* and *outfile* specify the same string, only one file is created and *stdout* and *stderr* point to the same descriptor.

Possible formats are

```
errfile file-name  
errfile format parameter-list
```

where *file-name* is a full or partial file specification. Only string data type is supported—no integer or floating data.

*Format* is a *cmdserv* allowed format (see above). *Parameter-list* is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of *%s* designators in the format string.

#### 4.2.5.7 Argument Directive

The *argument* directive tells *cmdservd* to copy the directive parameter(s) to the application's command line arguments in the order given. See *execve(2)* man page. Possible formats

```
argument argument-string  
argument format parameter-list
```

where *argument-string* is one full argument in string format. Only string data type is supported—no integer or floating data.

Format is a `cmdserv` allowed format (see above). `Parameter-list` is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of `%s` designators in the format string. Examples:

```
argument -x
argument inputfile.dat
argument %s-%s TEAM PLAYER
```

#### 4.2.5.8 Environment Directive

An environment directive tells `cmdservd` to define an environment variable in `cmdservd` process space. See *setenv(1)* and *putenv(3)* man pages. Environment variables can be used to generate the other application attributes, i.e., arguments, directory, file names. Environment variables also are inherited by the child process, and thus can be used to transmit data to the application.

In some cases, this method of transmitting input parameters to the child has an advantage over using the `argument` directive. Those situations include when security is an issue, because using UNIX can make arguments visible via the `ps` command.

Possible formats are

```
environment env-name env-value-string
environment env-name format parameter-list
```

where `env-name` is the environment variable name. `Env-value` string is the environment variable value. Only string data type is supported—no integer or floating data.

Format is a `cmdserv` allowed format (see above). `Parameter-list` is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of `%s` designators in the format string.

**Note:** Environment variables subsequently used in `executable` or `password` directives, which are affected by the client message, are not allowed. The command server daemon terminates the entry and does not run the specific application, because to do so would constitute a security hole. In other words, the client can not specify its own password nor its own executable file. Only the configuration file can do that.

Examples:

```
environment OPTION -x
environment SPORT BBall
environment TEAM SeattleSuperSonics
environment PLAYER Payton
environment TEAMPLAYER %s.%s TEAM PLAYER
```

### 4.2.5.9 File Directive

The `file` directive instructs `cmdservd` to create and open a new file to receive records. Records are written to the file via the `put` directive.

Possible formats are

```
file file-name  
file format parameter-list
```

where `file-name` is either a full or partial file specification. If a relative file name, the default directory is utilized as the starting point.

Format is a `cmdserv` allowed format (see above). `Parameter-list` is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of `%s` designators in the format string. Examples:

```
file /home/femis/user/evlog/10000745/e0/  
file /home/femis/user/evlog/%s/e%s/pf CASE EXER
```

In the first example, the `file` directive uses a full path specification involving no variables. The second example utilizes two variables `CASE` and `EXER`, assumed to be sent by the client.

A command server configuration file entry can utilize multiple `file` directives, in which case multiple files are created.

### 4.2.5.10 Put Directive

The `put` directive instructs `cmdservd` to copy one record into the file created and opened by the most recent `file` directive.

Possible formats are

```
put record-text  
put format parameter-list
```

where `record-text` is the actual and full record text to be copied into the currently opened file.

Format is a `cmdserv` allowed format (see above). `Parameter-list` is a list of environment variable names, which may be internal and/or external (client generated). The number of environments in the list must match the number of `%s` designators in the format string. Examples:

```
put "The quick brown fox jumped over the lazy dog."  
put %s-%s CASE EXER  
  
environment ANIMAL elephant.  
put "The quick brown fox jumped over the %s." ANIMAL
```

The first example copies a fixed static string into the file. The second utilizes a format string and two environment variables. The third example uses a quoted string as the format and one environment variable. The `ANIMAL` value could be provided in a message from the client.

#### 4.2.5.11 Allow Directive

A description of the `allow` directive is also included in `ACCESS` block documentation. Combinations of `allow` and `deny` can be used in `ACCESS` and `ENTRY` blocks to describe the permitted users of the command server.

Syntax of the `allow` directive is the keyword `allow`, followed by an IP address or subnet, followed by an optional subnet mask, followed by an optional comment.

Format of IP address and subnet mask currently is four decimal numbers, in the range 0-255, separated by decimal point. Allowed IP address elements are 0-255.

Allowed IP mask elements are 0, 128, 192, 224, 240, 248, 252, 254, and 255. Subnet mask must be in the format `255...xxx.0...`, where 255 can appear one, two or three times; 0 can appear one, two, or three times; and `xxx` (not 0 or 255) can appear only one time. Examples:

```
allow 0.0.0.0                # world  
allow 130.20.0.0 255.255.0.0 # pnl net  
allow 192.101.108.0255.255.255.0 # pnl-remote  
allow 130.20.92.131         # workstation  
allow 201.8.44.64 255      255.255.224      # subnet
```

#### 4.2.5.12 Deny Directive

A description of the `deny` directive is included in the `ACCESS` block documentation. Combinations of `allow` and `deny` can be used in `ACCESS` and `ENTRY` blocks to describe the permitted users of the command server.

Syntax of the `deny` directive is the keyword `allow`, followed by an IP address or subnet, followed by a subnet mask, followed by optional comments.

Format of IP address and subnet mask currently is four decimal numbers, in the range 0-255, separated by decimal point. Allowed IP address elements are 0-255.

Allowed IP mask elements are 0, 128, 192, 224, 240, 248, 252, 254, and 255. Subnet mask must be in the format 255...xxx.0..., where 255 can appear one, two or three times; 0 can appear one, two, or three times; and xxx (not 0 or 255) can appear only one time. Examples:

```
deny 0.0.0.0           # world
deny 196.104.8.0      # subnet
deny 130.20.92.87     # workstation
deny 201.8.44.32      255.255.255.224 # subnet
deny 201.8.44.96      255.255.255.224 # subnet
```

## 4.3 cmdserv – FEMIS Command Server Test Client (UNIX)

### 4.3.1 Synopsis

```
cmdserv [-v] [-h] [-D] [-u] [[IPaddr] | [hostname]] [port]
```

### 4.3.2 Availability

Program `cmdserv` is a test client for use with the FEMIS command server daemon `cmdservd`. The command server, test client, and related files are delivered in the FEMIS distribution tar file on magnetic tape or CD. The default locations for these files are `/home/femis/bin` and `/home/femis/etc` on the FEMIS UNIX data server.

### 4.3.3 Description

FEMIS utilizes remote command servers, executing on a UNIX host computer in order that PC workstation users can launch large mathematical model/simulation codes, which on the PCs either could not be run at all or would require an unreasonable amount of time and resources. These include the Evacuation SIMulation (ESIM), a module in the Oak Ridge Evacuation Modeling System (OREMS).

The command service consists of a client and server. The client runs on a Windows workstation. The server runs on UNIX and is capable of spawning processes at the request of a remote client.

This program is a client for use on the UNIX platform. Its purpose is mainly for testing the command server, for testing of new configuration file scripts, and for testing executables.

### 4.3.4 Options

The command server test client `-v` option produces a listing of current version information. Example:

```
virus% cmdserv -v
cmdserv version 1.0 - Wed Feb 14 14:41:00 PST 1996
```

The `cmdserv -h` option produces a help listing:

```
virus% cmdserv -h
usage: cmdserv [-hvD] [IPaddr | host] [port]
  -v      : display version information
  -h      : display help messages
  -D      : use unregistered service port (9015)
  Ipaddr  : host IP address, e.g., 130.20.92.87
  host    : server's host name, e.g., cemsun
  port    : protocol or service port, e.g., 9015
```

The `cmdserv -D` option turns on diagnostics.

Normally, the destination port is 9015, the standard service port for the FEMIS command server. Certain testing activities may require changing the `cmdserv` port number, thus the option to place it on the command line.

The destination host must be specified either as an IP address, or as a host name. One or the other must be specified, but not both. The local host can be designated as the command server daemon by including minus sign ( - ) in place of the IP address or host name. Examples:

```
virus% cmdserv locusts
virus% cmdserv virus
cemsun% cmdserv tcemsun
cemsun% cmdserv cemsun
virus% cmdserv -
virus% cmdserv 130.20.92.87
locusts% cmdserv 130.20.28.43
```

### 4.3.5 Installation

See the `cmdserved(1)` man page.

### 4.3.6 Protocol

See the `cmdserved(1)` man page.

### 4.3.7 Operation

Run the command service test client by entering `cmdserv`. `Cmdserv` first tries to connect with the command server daemon, `cmdserved`. Generally, any I/O error during execution of the test client will cause it to terminate. The possible errors during client operation are

```
cmdserv: create socket failed: PERROR
- Call to socket() library function to create a socket failed with the error
  indicated.

cmdserv: convert IP address failed: PERROR
- Call to inet_addr() library function failed with the error indicated.

cmdserv: HOST - unknown host: PERROR
- Call to gethostbyname() library function failed with the indicated error.

cmdserv: HOST-OR-IP - connect failed: PERROR
- The connect() library function call failed because of the indicated error.

cmdserv: HOST-OR-IP - can't get socket info: PERROR
- Call to getsockname() library function failed because of the indicated
  error.

cmdserv: read failed: PERROR
- Call to recv() library function to receive a message on a socket failed with
  the error indicated.

cmdserv: send failed: PERROR
- Call to send() library function to transmit a message on a socket failed
  with the error indicated.
```

where `HOST-OR-IP` will be either the destination host name or the destination IP address depending on how the command line was entered. And `PERROR` represents an error message returned from `perror()`.

Once `cmdserv` receives control from the shell, it opens a connection to the specified destination host and prompts for an action.

## Action

Prior to entering anything, wait for the server and client hello messages to be exchanged. `Cmdserv` displays two to three messages. Example:

## Received

```
<op:MISCINFO|
  program argv  :  cmdservd|
  program argc  :  1|
  current dir   :  /files0/home/larryg/femis/command/log|
  config file   :  \LNull\R|
  daemon uid    :  1033|
  getpeername   :  clen : 16|
  getpeername   :  gprc : 0|
  client port   :  2377|
  client host   :  hattrick.pnl.gov|
  client Ipadd  :  130.20.92.87|
  hwid number   :  82145C57|
  server key    :  \Lnull\R|
```

```
client key      : \Lnull\R|
process id     : 10332|
parent id      : 146>
```

### Received

```
<op:SVRHELLO|F2BBE247|*****|*****>
```

### Sending

```
<op:CLIHELLO|*****|*****|mo:alert test >
```

### Action

At this point, enter one of the following:

```
run X          : runs entry X from configuration file
status         : returns status of current application
kill           : kills the current application
```

After entering `run X`, `cmdserv` prompts for a password.

### Password

Either enter the password required by the configuration file or enter `Return`, if none is required. Also see the configuration file `cmdserv.conf(5)` man page.

`cmdserv` next prompts for any number of parameters. Parameters must be of the form `VARIABLE=VALUE`, where `VARIABLE` is the name of a variable in the command server, and `VALUE` is the value to be assigned.

**Note:** All values are string values. Numeric, integer, and floating point data are not supported in this implementation.

Once all parameters have been entered, type `return` or `^D`.

As soon as the command server processes the command and starts the application, it sends a message back to `cmdserv`, which is displayed:

### Received

```
<op:REPLY|rc:active TIMESTAMP PROCESS>
```

where `TIMESTAMP` is a 10 character time stamp, e.g., 9602071334, and `PROCESS` is the PID of the child process.

While the application is executing, entering status returns status of the application process. Once the application has terminated, the command server sends an alert message and `cmdserv` displays:

### Received

```
<op:ALERT|rc:finish TIMESTAMP PROCESS>
```

where `TIMESTAMP` and `PROCESS` are the same as above.

Now enter another command or exit via `^C` or `^D`.

## 4.3.8 Messages

Any of the possible command server daemon (`cmdservd`) error messages and reply messages can be received in the test client and thus be displayed on its standard output. See the `cmdservd(1)` man page.

## 4.3.9 Configuration File

See the `cmdserv.conf(5)` man page.

## 4.3.10 Service Port and Name

The `cmdserv` service port number currently is 9015. The short name is `femis-cmdserv` or `fxcmdserv`.

## 4.3.11 Files

Files utilized during the installation and execution of the FEMIS command server include

<code>/home/femis/bin/cmdservd</code>	daemon executable
<code>/home/femis/etc/cmdserv.conf</code>	configuration file
<code>/home/femis/bin/cmdserv</code>	test client (UNIX)
<code>/etc/services</code>	service port numbers
<code>/etc/inetd.conf</code>	internet daemon config