OSF DCE SIG **H. Melman (OSF)**
Request For Comments: 30.0 **January 1993**

DCE 1.1 ADMINISTRATIVE IMPROVEMENTS: REQUIREMENTS FOR DCE SHELL

1. INTRODUCTION

A goal of DCE 1.1 is to make DCE easier to administer. One of the current problems with DCE is the fact that there are many different control programs for different components, with different syntaxes. Often, more than one is needed to perform certain operations, such as creating a CDS entry with an ACL. This RFC discusses the need for a new program, known as the *DCE shell*, providing an administrative user interface that can augment the use of the existing programs, at least for commonly performed tasks.

2. GOALS

The following are goals and guidelines for the DCE shell.

(a) The major goal is to simplify the administration of a DCE cell by creating a single powerful and consistent administrative user interface.

(b) The intended users of the DCE shell are DCE system administrators. DCE developers will probably see benefit in using it, however, they are not the primary consumers.

(c) Emphasis in scheduling development resources must be placed on common everyday administrative duties rather than more esoteric configuration management. Administrators will get more use from a simpler way to add a new user or to modify a CDS entry and its ACL, rather than a new way to create a registry or to initialize a new CDS namespace.

(d) The DCE shell will only be used to administer DCE itself, it will *not* be used to administer applications that are written with DCE.

(e) The DCE shell will augment use of existing control programs, it will *not* replace them. Existing control programs will continue to be installed with DCE, and their use will probably remain required for some administrative operations.

(f) Commands must be remotely invocable. This is currently true for most existing control program operations. The extension of those commands that are not remotely invocable is desired.

(g) The DCE shell must be portable to a variety of different platforms and management situations. Venders must be able to integrate it into their own management systems.

(h) A graphical user interface (GUI) is desirable, but a command line interface is: more portable to a wider variety of platforms, scriptable, and usable over telnet lines and on machines not equipped with GUI's. Therefore, the command line interface is more important for the first release of the DCE shell.

(i) This is not DME. The DCE shell is not meant to replace the need for DME, but rather to substitute for its absence in a very limited manner. Nothing in the design of the DCE shell should preclude the use of DME in the future.

3. REQUIREMENTS

This section identifies features which must be included in the first release of the DCE shell in order for it to meet the goals stated above. In several instances items of lower priority are described, as they are still desirable.

3.1. Existing Control Program Functionality

The DCE shell must be able to perform some or all of the functions currently performed by the following DCE control programs:

(a) `rpccp`
(b) `cdscp`
(c) `acl_edit`
(d) `rgy_edit`
(e) `dtscp`
(f) `sec_admin`

The goal is to have the DCE shell perform as many functions as are possible, realizing that there are constraints on available implementation time.[1] The DCE shell must handle those operations that are most common with the goal that commands which cross component boundaries (e.g., adding a CDS object with an ACL) can be implemented as well (i.e., in order to add a CDS object with an ACL, the DCE shell must be able to both add a CDS object, and create an ACL). The priority is that the DCE shell be able to handle the day-to-day administrative operations of a DCE cell; however, the ability to also perform DCE configuration would be very beneficial, as this is the area of administration most used during development.

By duplicating functionality in the DCE shell, caution must be used to ensure that any changes needed in either an existing control program or in the DCE shell, must be reflected in the other. It is expected that over time, use of the existing control programs will diminish and be phased out. Therefore, new features to the administrative interfaces must be added to the DCE shell as opposed to existing control programs.

3.1.1. REQT: Stop servers

Where relevant, the DCE shell must be able to stop all core daemon and clerk processes (i.e., `rpcd`, `sec_clientd`, `cdsd`, `dtsd`, `secd`, `cdsadv` and `cdsclerk`) that can currently be stopped by a command from a control program.

3.1.2. REQT: Show/add/modify/delete data

All of the control programs manipulate different data.[2] The general operations are show, add, and delete. Often, but not always, modify is available as well. The DCE shell must be able to completely replace an existing control program for manipulating any data it is capable of manipulating. That is to say, the DCE shell might not be able to deal with all data, but for any data that it is capable of dealing with, it must support at least all existing functions on that data.

For example, `rpccp` provides commands that operate on server entries in the namespace. These commands let server entries be added, viewed, and removed, but there is no command to modify (edit) a server entry. If operations on server entries are added to the DCE shell, it must at least be able to add, display, and delete server entries (the current functionality). The ability to modify server entries would be a useful feature in the DCE shell, but is not required.

The following is a list of the different types of data that are manipulated via current control programs. The DCE shell must be able to operate on as many of these as possible. The list is presented grouped by current control program functionality. Those items marked with an * are of highest priority and must be included in the first revision of the DCE shell.

(a) `rpccp`
    * (i) Server Entries
    * (ii) Groups
    * (iii) Profiles
    * (iv) Endpoint Mappings

---

1. See Appendix A for a list of all commands supported by these control programs. See Appendix B for a list of the line counts of the modules in these control programs.

2. A better word than data would be objects; however, object has a special meaning in the directory services, so its use will be avoided. Examples of data are: groups, profiles, directories, objects, PGO's, accounts, ACL's, keys, etc.

(b) `cdscp`
  * (i) Directories
  * (ii) Objects
  * (iii) Softlinks
  (iv) Clearinghouses
  (v) Replicas
  (vi) Child Pointers
  (vii) Cache Entries

(c) `acl_edit`
  * (i) Access Control Lists

(d) `rgy_edit`
  * (i) Principals
  * (ii) Groups
  * (iii) Organizations
  * (iv) Accounts
  (v) Keys

(e) `dtscp`
  (i) Set/Show Various Server Attributes

3.1.3. REQT: Show status

Many of the existing control programs can display current status of running processes (clerks and servers). The DCE shell must be able to do this as well via existing interfaces. Better and more consistent status reporting would be desirable, but would probably require changes to servers, and therefore is beyond the scope of this project. The Serviceability Requirements Document [RFC 24] examines this in more detail. The DCE shell will address this in as much as it will perform all serviceability administrative functions, including remotely turning on and off logging.

3.1.4. REQT: Remote operations

The DCE shell must continue to support the ability to perform operations remotely wherever that ability currently exists. It is desirable (but not required) to extend certain operations that are currently limited to local use to be remotely invocable.

3.2. Usability

While duplicating existing control program functionality within a single program will make administrative operations somewhat easier to perform, there are other requirements to be met to ensure that this single program is easy to use.

3.2.1. REQT: Common syntax for all functions

Currently each control program has its own syntax (although some are similar). The differences are sometimes vast; for example, compare the user interface of `cdscp` to that of `acl_edit`. The DCE shell must have one syntax for dealing with all its functionality. For example, the deletion of a principal must be invoked in a very similar manner as the deletion of a profile.

3.2.2. REQT: Help

The DCE shell must provide interactive on-line help. Help for each command must be available and easily accessible. A minimum is to supply usage information for each command and option. Better would be to supply on-line documentation for how and when to use a command or option, which could help a user through a process.

3.2.3. REQT: Provide command line recall/editing

The DCE shell must provide some form of `ksh`-like command line recall and editing capability. This is not difficult to implement[3] and greatly increases usability. In fact, command line editing is included in [POSIX.2].

Input to the DCE shell must be available in two forms: as either a single command from a shell prompt,[4] or from within a DCE shell command parser with a DCE shell prompt. This is how several existing control programs behave (e.g., `cdscp`, `rpccp`, and `dtscp`).

3.2.4. REQT: Scriptability

All functions of the DCE shell must be scriptable. Some existing control programs (e.g., `acl_edit`) currently limit this to a shell script that invokes a control program with a single command at a time. However, as experience with `dce_config` and `cdscp` has shown, the overhead of invoking a control program for every command can be quite large. The DCE shell must allow scripts to be input via standard input or a specified file as several control programs currently allow (e.g., `cdscp` and `rpccp`).[5]

Having command line recall implies the existence of a history mechanism in the DCE shell. The DCE shell must provide access to this history so that scripts can be created from it. The ability to display the current history log, or to send it to a file would be sufficient. Any other mechanisms to make script writing and testing easier should be implemented as well. These include a no execution mode and an echo command line mode.[6] Also included are the ability to redirect output to files, and sufficient error handling mechanism to allow for efficient debugging of scripts.

3.3. New Functionality

3.3.1. REQT: Administer serviceability and auditing

The DCE shell must be a single administration program that can perform all administrative functions for serviceability and auditing. These are new components being added to DCE 1.1, see [RFC 24] and [RFC 25] for details on these services. Whatever the administrative requirements for these are components are, the implementation will have to be created from scratch, and should be in the DCE shell as opposed to any place else. The requirements will at least include the ability to set logging levels (including no logging) and the ability to view generated logs.

3.3.2. REQT: Programmability

The scripts that the existing control programs accept are limited to just the commands that perform DCE operations. There are no variables, expression evaluation, or control structures supported in any of them. In order to use such facilities, shell scripts are written that invoke the control programs, the best example of such a script being `dce_config`. As described above, this can be flexible, but often at the performance cost of multiple invocations of the control programs. In addition, shell scripts are not portable to non-Unix environments, and even within Unix environments there can be problems (e.g., the OSF/1 `/bin/sh` does not recognize the `return` command).

The DCE shell must offer internal variables, expression evaluation, and control structures (`if`, `while`, `for`, etc.) to its users. These features will make the DCE shell a programmable utility, thereby allowing for the creation of powerful portable scripts written in the DCE shell language. A DCE shell script could be run on heterogeneous machines in the same cell, making some system management tasks easier.

One commonly requested capability is to perform the same operation on several different objects (either data or servers). Examples of this type of iterative operation are: setting ACL's on several CDS entries and getting status from all `dtsd`'s in a cell. The method the DCE shell uses of handling information is important in the implementation of this second example. It would be most useful if the output of one command (in this case a command that lists all `dtsd`'s in a cell in an addressible manner, probably by looking in the namespace) could be used as input to another command (in this case `show-status`).[7]

---

3. In fact, OSF has a version of readline that can be used without copyright problems.

4. It is preferable if the DCE shell reads a line of input rather than input separated into words. This will allow the use of simpler shell quoting mechanisms when creating input lines to the DCE shell.

5. It is common practice for programs to accept a `-f` switch with an argument specifying a filename to read from, or if the argument is absent or is a `-` then to read from standard input. See `awk` and `expect` for examples.

6. As is provided respectively by the `-n` and `-v` switches of the `csh`.

7. An implementation of this feature is command substitution (via backquotes) as found in the Bourne and Korn Shell, combined with some iterative command such as Lisp's `mapcar` or the C Shell's `foreach` command.

The following is an example of possible syntax used to get status from all dtsd's in a cell. It assumes the existence of a listall command that takes an argument of a server name and returns a list identifying all instances of that server in the cell, as well as a show-status command. Square brackets are used to delimit command substitution.

```
foreach server [listall dtsd]
    show-status $server
```

This is not meant to define syntax used in the DCE shell, it is meant solely to describe how control structures can be used to make performing some common tasks easier (or even possible).

3.3.3. REQT: Extensibility

The DCE shell language must provide the capability to create new procedures, implemented as DCE shell scripts. This will make the administrative functions of the DCE shell easily extensible from a user level. By implementing the functions in the existing control programs as low level DCE shell functions, and allowing for the creation of higher level procedures, the DCE shell will be a very flexible utility. Given the basic routines to add principals, accounts, and objects in the namespace, an add user command could be created that does all of these things with one command. Created commands must be available via scripts so that they can be saved and reused. The simplest implementation would require the use of a command similar to the C Shell's source command, but something more advanced is easily envisioned such as searching for a builtin command and then looking in a PATH variable for scripts. It is expected that scripts implementing commands such as the add user command above would be shipped with the DCE shell.

By allowing end user extensibility in this manner, development of the DCE shell can be ongoing. Once the base commands are implemented, various development groups can work towards adding new features. For example, while hierarchical cells are being developed, any base commands could be built into the DCE shell (they will have to be somewhere), and then new commands could be added for various configurations, such as rename cell and create new child cell. Also, testing groups can be developing commands to help in new kinds of configurations that would be needed to run tests such as configure client, configure replicated registry, move client to another cell, etc.

Finally, other development groups could add base commands that could be incorporated into other commands. For example, if DFS commands are added to the DCE shell, then the add user command could create a home directory and fileset for the new user. This currently involves about four smaller steps, such as creating an aggregate if it doesn't already exist, creating a fileset, doing an export, etc. In addition, the add user command could be extended to set up backups and replication.

3.3.4. REQT: Bind to different servers or data

The DCE shell must be able to operate on different data and servers within a single session. The current limitation in acl_edit in this regard must not be duplicated in the DCE shell. There must be some scoping or selection mechanism to identify what data or server a command will operate on. It is a requirement that some commands be able to identify a particular server to operate on. Commands that show current status or initiate a replication update, must be selectable on a per server basis.

3.3.5. REQT: Shell escape

The DCE shell must provide the ability to escape to a command shell[8] in order to execute system commands. This will allow for the manipulation of local files and other operating system functions.

An example of the usefulness of this feature is a backup command that would use operating system commands to backup DCE system files such as CDS checkpoint files. A better example is backing up the security registry files, which involves placing the master registry server in a maintenance mode via the current sec_admin command state -maintenance and then using local operating system commands to backup files in the local filesystem.

---

8. Lacking a better term, command shell will be used in the traditional sense of the word shell (i.e., shells such as the Borne Shell, Korn Shell, and C Shell).

This process could be turned into a DCE shell script or user created command, and could also be combined with backing up information from other DCE components.

Commands given while in the command shell need not be stored in the history mechanism, but there must be a way to invoke a command shell with a single command line which is stored in the DCE shell history mechanism. This will allow DCE shell scripts to be able to invoke command shell operations or scripts. Note that the use of a shell escape in DCE shell scripts can limit their portability, particularly to non-Unix environments.

3.3.6. REQT: Administration of little files

Currently there are several files kept on local systems with DCE configuration information. The best example of such a file is `dce_cf.db` which contains information about the cell that the machine is in. In order to change the default cell of a client machine, this file would have to be edited on the client machine. The DCE shell must be able to modify this file for this type of operation (change default cell). This functionality could be limited to the local case, and access to the file via a shell escape would be sufficient.

3.3.7. REQT: DCE ping

The DCE shell must be able to ping a system to see if services are receiving rpc requests. This is a simple interface to `rpc_mgmt_is_server_listening(3rpc)`.

3.3.8. REQT: Security

Access to the interfaces implemented in the DCE shell must be secured in a consistent manner across all DCE shell commands.

3.3.9. REQT: Meet Other DCE 1.1 Goals

The DCE shell must meet the other DCE 1.1 cross component goals. These include but are not limited to the following:

(a) Quality

The DCE shell must meet the DCE 1.1 goals for maximum defects per KLOC,[9] and for testing coverage.

(b) Serviceability and Auditing

The DCE serviceability functions for logging and messaging must be used. Any functions matching auditing event criteria that require the generation of an auditing record, must do so.

(c) Code Clean-Up

The DCE Code Clean-Up requirements for producing [ANSI C] compliant code, requiring the use of standardized library calls, and isolating machine dependencies must be met. (See [RFC 17] for details.)

(d) Internationalization

As the DCE shell will present an interface to a user, all messages it produces are subject to internationalization requirements. See [RFC 23] for details.

3.4. Lower Priority Requirements

3.4.1. REQT: Support for other DCE administrative functions

The motivation for a single DCE shell control program came mostly from experiences with administering DCE cells consisting of core components only, and as such, only core component functionality has been considered in great detail. Adding DFS and/or GDS support would certainly be useful in those cells in which these are run.

GDS administration is currently performed by a menu driven control program implemented using the curses interface. While it does offer menus (usually considered an improvement over a simple command line interface), it is

9. 1,000 lines of source code, not counting whitespace or comments.

still often difficult to use. It also possesses scripting capabilities, but scripts are quite difficult to generate as they involve creating full screen images. An item that is being considered for DCE 1.1 development is a command line interface to GDS. If this is done, then it should be as part of the DCE shell, or at least as a clone of the DCE shell meeting the same usability requirements as described above in the same manner as the DCE shell (i.e., the syntax must be the same).

DFS administration is currently performed by several command suites such as the `bos`, `cm`, and `fts` commands. They all take subcommands as arguments but are limited to command line input (i.e., you cannot enter a `bos` shell and enter `bos` commands as `cdscp` allows). There are about ten of these command suites in use. Currently the OSF development group does not use shell scripts to perform common functions requiring more than one command, however that is easily envisioned. In general, DFS is perfectly suited to the approach of implementing base routines in the DCE shell and then using a builtin language to extend the commands. The issue is whether there is time to implement this in DCE 1.1.

There are other core component administrative functions that have not been mentioned so far either. Particularly the security commands `passwd_import` and `passwd_export`. It is not clear that the DCE shell could affect the ease of use of the commands, as they require no user interaction. It may suffice to be able to invoke these commands via the shell escape mechanism described above. Access to these commands via the DCE shell could be useful for some commands such as including execution of `passwd_export` whenever a new user is added to the system.

3.4.2. REQT: Graphical user interface

It has already been stated that a GUI is less important than a command line interface for the first release of the DCE shell. However, a GUI of some form would certainly be a valuable addition to the DCE shell. What form should it take? It would be most desirable if a diagram of the running cell could be shown identifying all hosts and servers and allowing a user to click on an icon to display status or to select it for a specific action. It is believed that this type of interface will be difficult to develop in the available timeframe. However, design decisions that could allow for the future integration of such an interface should be made whenever possible.[10]

A more practical GUI (to implement) would have a window used to display information, (e.g., lists of entries, principal, accounts, entries, directories) with pull down menus allowing actions. This would be very similar to the existing `cdsbrowser` and `cdsedit`.[11] The display window would contain information that could then be selected for another operation. For example, a command that lists all `cdsd`'s running in the cell, would show a list of them in the display window. One could then be selected with the mouse, and then status for that server could be obtained with another command.

3.4.3. REQT: Command completion

Command completion would allow for both commands and perhaps some arguments (e.g., names in the namespace) to be completed in the same manner as `csh` and `ksh` allow for filename completion. This would also improve usability as is evident by the use that filename completion gets from the shells. It would be useful to implement, but other items are higher priority. Whether the namespace completion could be implemented with sufficient performance to make it useful is unknown. The extensibility requirements described above could make implementation of command completion difficult, particularly where user written commands take arguments which are to be subject to completion.

---

10. For example, often command line interfaces and GUI's have very different paradigms for their commands. GUI's usually are object oriented where some object is displayed in a window and actions are invoked on it, whereas command line interfaces are verb based where an action is invoked with an object as its argument.

11. The `cdsedit` program is a namespace browser that provides update capability in addition to browsing, as well as the ability to operate on ACL's on CDS objects. It was developed by the University of Massachusetts at Amherst, and has been demonstrated to OSF.

4. ISSUES

The items in this section require some further investigation or decision on their inclusion or exclusion from DCE 1.1 requirements.

4.1. Starting Servers

A useful feature of the control program would be the ability to start a server instance on a node. Currently most control programs do not support such a function. It would not be difficult to create such a command in the DCE shell, but its advantage over starting processes from a shell is not terribly clear, unless the DCE shell also has the capability to start servers on other hosts. An implementation of this function would probably require `rpcd` to be able to spawn processes (similarly to `inetd`), which is an item currently under consideration for DCE 1.1

4.2. Extending Existing Services

It is expected that in the course of DCE shell development, other inconsistencies between services (e.g., security models, server binding information, etc.) will be discovered that require a change or extension to an existing service. The issue is what should be done in these cases? By definition the DCE 1.1 release has the requirement that there be no incompatible protocol changes made. Given this rule, all that can be said is that each discovered change or extension must be handled on a case-by-case basis.

4.3. Selecting a Particular Server

For some commands, it is desirable to be able to identify a particular server for that command to be executed on. Examples include obtaining the status of a server, and forcing a replication update to occur. A manner of identifying a particular server on a particular host must be developed. The following are options:

(a) UUID's
(b) String Bindings
(c) Namespace Conventions

4.4. Intercell Support

Are there any special requirements for supporting intercell configurations? Should the DCE shell be able to set up the trust relationships for intercell work? Should it be able to do this for both cells involved? Is there any other support required?

4.5. Use of Tcl

Tcl is an interpreter for a Tool Command Language developed by John Ousterhout at the University of California at Berkley [TCL]. It is a library that is meant to be linked into a tool program which provides a programmable language for the tool. It would provide an implementation of the programmability requirements described above. It also allows for the creation of new procedures which would provide an implementation of the extensibility requirements described above. In addition, Tcl provides certain features, such as detailed error handling, that meet some of the scriptability requirements. Using Tcl would require that the DCE base commands be implemented as Tcl base commands, which is essentially registering them with an interpreter and ensuring that uniform return codes and values are used.

The use of Tcl could make development of a fully functional DCE shell much easier. In fact, many of the requirements described in this document came about after having first learned about Tcl. If starting from scratch, many of the programmability and extensibility requirements will be difficult to meet in the available timeframe. Using Tcl, allows the requirements to be met with minimal work,[12] so that most of the time can be spent developing the DCE commands; and it is expected that much of this code can be taken from the existing control programs with few changes.

In order to use Tcl, it will have to meet the same requirements of any code that OSF obtains from other sources. Here is a list:

12. In fact a DCE shell prototype already exists, using Tcl, meeting these requirements and implementing a DCE ping command.

(a) Intellectual Property.

OSF needs the right to work with the code as needed. This applies to copyright (and copyleft) restrictions. The best thing is for OSF to have unlimited right to use and distribute the code in whatever form we want, without complying with any rules with ‘‘indemnification’’, meaning the supplier will defend us against intellectual property claims. (Our licensees ask us to do this for them.)

(b) Functionality Limitations.

It must meet the full functionality requirements of other DCE code. If it does not, we must add this to the code. Certainly the code will have to meet our coding guidelines, defect density, test coverage, reliability, etc. goals, just as if we'd written it ourselves. The current list of areas with known functional limitation concerns in Tcl code are:[13]

(i) Internationalization

Currently Tcl is 8-bit clean with perhaps a few lingering exceptions. Unfortunately, there is little other internationalization support. The messaging support should not be difficult to add, but some codeset issues might be. The only datatype used in Tcl is the string. All commands return their results as strings. Of particular concern, are places in the code where the size of a character is assumed to be one byte. There are a few codeset dependent issues (e.g., case changing) but these should not be difficult to correct.

(ii) Thread-safeness

Tcl should be pretty clean in this respect. The fact that the interpreter is highly recursive forces it to be re-entrant; there aren't any global variables, except for environ, etc. However, it isn't safe to have multiple threads executing in the same interpreter at the same time, and trying to modify Tcl to support this would be pretty difficult. Thus either you can have separate threads each with its own interpreter, or you have to wrap each Tcl call with synchronization primitives to ensure mutual exclusion.

(c) Version Issues

There must be a strategy to stay in synch with new versions of Tcl as they become available. If we diverge from the main line (more than just compiling with CMA), then this could more than trivial. These problems have been solved before, in fact, DCE currently faces this issue with Kerberos code.

APPENDIX A. EXISTING CONTROL PROGRAM COMMANDS

The following are the commands supported by various control programs as found in the DCE 1.0.1 reference pages. The fact that they are listed differently (sometimes alphabetically, sometimes by function) indicates that there is a consistency problem.

This list is not intended to be a list of all the things that the DCE shell must do, but rather as a starting point for what it could do. Note that commands like `help`, `quit`, and `exit` are *not* listed.

A.1. rpccp

(a) All entries

```
add entry                  show entry
remove entry
```

(b) Server Entry

```
export                     show server
import                     unexport
```

13. The status in these items comes from email discussions with John Oustehout.

(c) Groups

`add member`	`remove member`
`remove group`	`show group`

(d) Profiles

`add element`	`remove profile`
`remove element`	`show profile`

(e) Endpoint map

`remove mapping`	`show mapping`

A.2. cdscp

`add directory`	`list object`
`add object`	`remove directory`
`clear cached server`	`remove link`
`clear clearinghouse`	`remove object`
`create child`	`set cdscp confidence`
`create clearinghouse`	`set cdscp preferred clearinghouse`
`create directory`	`set directory`
`create link`	`set directory to new epoch`
`create object`	`set directory to skulk`
`create replica`	`set link`
`define cached server`	`set object`
`delete child`	`show cached clearinghouse`
`delete clearinghouse`	`show cached server`
`delete directory`	`show cdscp confidence`
`delete link`	`show cdscp preferred clearinghouse`
`delete object`	`show cell`
`delete replica`	`show child`
`disable clerk`	`show clearinghouse`
`disable server`	`show clerk`
`dump clerk cache`	`show directory`
`list child`	`show link`
`list clearinghouse`	`show object`
`list directory`	`show replica`
`list link`	`show server`

A.3. acl_edit

`abort`	`kill_entries`
`assign`	`list`
`cell`	`modify`
`commit`	`permissions`
`delete`	`purge`
`exit`	`substitute`
`get_access`	`test_access`
`help`	

A.4. rgy_edit

(a) PGO’s

`view`	`member`
`add`	`delete`
`change`	`adopt`

(b) Accounts

```
view                          delete
add                           cell
change
```

(c) Key Mgmt

```
ktadd                         ktdelete
ktlist
```

(d) Misc

```
domain                        auth_policy
site                          defaults
properties                    login
policy                        scope
```

(e) Local Registry

```
view                          purge
delete                        properties
```

A.5. dtscp

```
advertise                     set server principal name
change epoch                  set servers required
change time                   set synchronization hold down
create type clerk             set global set timeout
create type server            show all
delete                        show all characteristics
disable                       show all counters
enable                        show all status
enable set clock              show {attribute-name}
set check interval            show global servers
set courier role              show local servers
set error tolerance           synchronize
set local set timeout         synchronize set clock
set maximum inaccuracy        unadvertise
set query attempts            update
set server entry name
```

A.6. sec_admin

```
become                        lrep
change_master                 master_key
delrep                        monitor
destroy                       site
info                          state
initrep                       stop
```

APPENDIX B. Existing Control Program Sizes

This appendix lists for each existing control program, each source file and its size as given by a count of the number of lines in each file. This is not KLOC but rather the output of the `wc` command.

B.1. rpccp

```
  80 cp_help.c
3238 rpccp.c
3318 total
```

B.2. cdscp Including msg, prs, and parser_aid

```
 232 accum_cmd.c
  74 argv.c
  99 attrgroup_actions.c
 236 attributes_list.c
 812 attrname_actions_cds.c
  57 baddest.c
 131 cdscp_help.c
  88 cmdblk_init.c
  97 confidence.c
 211 cp_cell.c
1489 cprpc_clerk.c
 451 cprpc_emit_clerk.c
 452 cprpc_emit_server.c
1180 cprpc_server.c
1130 cprpc_util.c
 238 default_ent.c
 132 dnscp_args.c
 650 dnscp_credir.c
 351 dnscp_crekid.c
 120 dnscp_creobj.c
 105 dnscp_cresl.c
 275 dnscp_delkid.c
  79 dnscp_delobj.c
  79 dnscp_delsl.c
 425 dnscp_help.c
  93 dnscp_replica.c
  81 dnsinitflagstat.c
 228 dumpclerkcachecmd.c
 374 emit_cds_values.c
  58 emit_chdir_line.c
  89 emit_chdir_lines.c
  79 emit_exp_line.c
 103 emit_ext_line.c
  60 emit_group_line.c
  86 emit_member_lines.c
  61 emit_myname_line.c
 397 emit_one_line.c
  65 emit_parentcts_line.c
  69 emit_parentptr_lines.c
  52 emit_pptimeout_lines.c
 374 emit_ref_hdr.c
  65 emit_repchcts_line.c
  60 emit_repchname_line.c
  80 emit_replicapointer_lines.c
  65 emit_reptype_line.c
  64 emit_timeout_lines.c
  98 emit_tower_lines.c
 104 emit_towerset_lines.c
 270 entities_actions_cds.c
 279 entities_list.c
 617 err_list.c
 278 err_report.c
 140 exec_docmd.c
```

```
 235 exec_mgmt_cmd_cds.c
 192 failure.c
 242 filespec_actions.c
  54 find_base_name.c
  62 find_base_name_unix.c
 120 fmt_uid_for_print.c
 101 fopen2.c
 124 format.c
  89 format_chstate_for_print.c
  54 format_classversion_for_print.c
  84 format_confidence_for_print.c
  85 format_convergence_for_print.c
 159 format_fullname_for_print.c
  85 format_inch_for_print.c
  84 format_kindofent_for_print.c
  20 format_nodeid_for_print.c
  72 format_piboolstatus_for_print.c
  86 format_repstate_for_print.c
  90 format_reptyp_for_print.c
  73 format_simplename_for_print.c
  61 format_timestamp_for_print.c
  23 format_timestamp_with_time.c
 286 format_tower.c
  95 format_utcabs_for_print.c
 167 get_command.c
  73 hash.c
  39 islatin1print.c
 231 load_msg_file.c
 212 load_parse_file.c
 654 logging.c
 560 mgmt_main.c
 262 mgmt_parsing.c
1044 msg.c
 354 msgfil.c
 457 msgfnd.c
 190 msgrea.c
 348 names_actions_cds.c
1528 op_list.c
  88 operator_actions.c
 124 parse_command.c
 232 parser_aid_2.c
 259 parsing_support.c
 142 preferred_clh.c
  59 pretend_not_seen.c
 125 process_commands.c
3576 prs.c
2114 prscmd.c
 354 prsfil.c
 399 prsrea.c
 300 qarith_lib.c
 141 recognize_filespec.c
 126 replica_actions.c
 225 replica_support.c
 197 report_parse_error.c
  54 savemajorminor.c
```

```
    114 seconds_to_vmsdeltatime.c
    179 spaces.c
    110 spawn_ncl.c
    149 startup_file.c
     84 stubs.c
     17 symsub_actions.c
    244 time_actions.c
     59 validate_tower.c
    910 visit_cds.c
    153 with.c
    801 cds.cdt
     32 parser_aid_2.cdt
    744 mgmt_msg_cds.txt
  34388 total
```

B.3. acl_edit

```
    237 acl_edit.c
     46 acl_edit.h
   2218 deif.c
    296 deif.h
    118 deif_err.c
   1408 tty.c
     56 tty_base.h
   4379 total
```

B.4. rgy_edit

```
    217 eif.c
    160 eif.h
    637 eif_acct.c
     94 eif_acct.h
    717 eif_def.c
    117 eif_def.h
    799 eif_pgo.c
    125 eif_pgo.h
    411 eif_prop.c
    119 eif_prop.h
    357 eif_site.c
    120 eif_site.h
     30 eif_util.c
     44 eif_util.h
   8853 ek_db.c
    679 ek_db.h
     30 ek_error.c
     32 ek_error.h
     83 ek_glob.h
    391 ek_site.c
     70 ek_site.h
    181 ek_types.h
   1992 ek_util.c
    597 ek_util.h
    438 form.c
    223 form.h
   1734 tty.c
   2126 tty_acct.c
     67 tty_acct.h
```

```
   38 tty_base.h
  184 tty_conf.c
  213 tty_conf.h
  283 tty_def.c
   45 tty_def.h
   30 tty_err.c
  125 tty_err.h
  793 tty_io.c
  261 tty_io.h
  328 tty_keytab.c
   42 tty_keytab.h
 2163 tty_pgo.c
   85 tty_pgo.h
 1139 tty_prop.c
  110 tty_prop.h
  234 tty_util.c
  115 tty_util.h
27631 total
```

B.5. dtscp Including libprs_threaded.a

```
 151 allocate_mem.c
2902 dtsscp.c
 101 dtsscp_ultrix.c
2222 prscmd.c
 358 prspre.h
 412 prsrea.c
 849 dtsscp.cdt
6995 total
```

B.6. sec_admin

Note that this file does not include all the replicated security functions.

```
931 sec_admin.c
931 total
```

B.7. libsecutil.a Included in acl_edit, rgy_edit, and sec_admin

```
203 args.c
338 commands.c
171 auth_test_util.c
712 total
```

REFERENCES

[PORTG] DCE Porting and Testing Guide, Open Software Foundation, 1992.

[POSIX.2] IEEE Std 1003.2 - 1988, Portable Operating System Interface — Shells and Utilities

[RFC 17] H. Melman, ‘‘DCE Code Clean-Up Requirements’’, December 1992.

[RFC 23] R. Mackey, ‘‘DCE 1.1 Internationalization Guide’’.

[RFC 24] R. Salz, ‘‘DCE 1.1 Serviceability Proposal’’, November 1992.

[RFC 25] E. McDermott, ‘‘DCE Auditing Design and Strategy’’, December 1992.

[TCL] Ousterhout, J. ‘‘Tcl: An Embeddable command Language.’’ Proc. USENIX Winter Conference, January 1990, pp. 133-146.

AUTHOR'S ADDRESS

Howard Melman
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: melman@osf.org
Telephone: +1-617-621-8989