DCECP FUNCTIONAL SPECIFICATION

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 must be used to perform certain operations, such as adding a user to a cell. This Functional Specification describes a new program, known as the *DCE Control Program*, abbreviated `dcecp`, providing an administrative user interface that will augment the use of the existing programs, at least for commonly performed tasks.

While the licensees and users of DCE 1.0 have found it to be a useful distributed computing platform, they have also found it to be a difficult environment to administer. The feedback we have heard in this area has made the simplification of DCE administration an important goal of DCE 1.1.

The fact that DCE is composed of several different components originating from different vendors, is readily apparent when viewing the administrative tools. Many of the difficulties in administering DCE arise from the need to use many different tools that behave inconsistently and operate at a very low level relative to administrative **tasks**[1] (e.g., add a user to a cell, configure a client machine, backup, etc.). This functional specification describes a new administrative tool known as `dcecp` which can perform many of the operations of existing control programs via a new consistent command-line interface. It is expected that DCE **administrators** will use `dcecp` instead of the existing **control programs** to perform most daily operations.

In DCE 1.1, `dcecp` will include functionality from the following control programs: `rpccp`, `cdscp`, `rgy_edit`, `acl_edit`, `dtscp`, and `sec_admin`. That is, it will be able to manipulate data (e.g., directories, entries, groups, principals, accounts, ACL's, etc.) stored in the various databases (e.g., the registry, clearinghouses, ACL managers, etc.). It will also be able to perform certain management operations (e.g, server shutdown, ping, get status, etc.). An important feature of `dcecp` is that it will provide a script language including but not limited to: variables, control structures, and the ability to create new procedures. This will allow administrators to create their own task **commands**. The first version of `dcecp` will include several such task commands implemented as `dcecp` scripts.

The `dcecp` Language will be based on **Tcl** (pronounced ''tickle''), the embeddable Tool Control Language developed at the University of California at Berkeley. The `dcecp` commands will be implemented as Tcl commands in a Tcl interpreter. This document assumes knowledge of Tcl and to a limited extent, its syntax.

1.1. Changes Since Last Publication

The following changes have been made since this document was previously published in [RFC 42.2]. They are given in the order they are found in this document.

(a) The `organization modify` command does not support the `-orgid` or `-uuid` option any more. These may not be modified. The text of `principal modify` and `group modify` were updated to state this explicitly as well. They did not accept attribute options for these but the text implied that the attributes could be listed in an attribute list.

(b) The options allowed to the `rpcprofile remove` command have been changed. A `-priority` is supported. Also the values of the options have been changed to single values and not lists.

---

1. This **boldface** convention is used to introduce terminology the first time it appears in this paper. See the *Terminology* section, below, for definitions.

(c) The specification of the `-tree` option has been expanded to say that the `-replica` and `-clearinghouse` options are not allowed if `-tree` is used.

(d) The description of the `directory merge` command now explicitly states that the `-clearinghouse` option applies only to newly created directories.

(e) The description of `link create` has been improved and an example added. The `-linkto` option is no longer required, though the value of the `CDS_LinkTarget` attribute is. Also the format of the `CDS_LinkTimeout` attribute has been specified, both as an attribute list, and as a short format for the `-timeout` option.

(f) The `registry modify` command's description was corrected. The command takes no argument and always contacts the master registry.

(g) The `registry show` command's description was corrected. The optional argument is the name of a registry replica to contact.

(h) The description of the `audevents show` command was made more precise. Also command outputs the event class name, not the event class number.

(i) The argument given to an `acl` command is either a name or a list of a string binding and a residual name.

(j) The description of lists of lists was improved.

(k) The `audfilter` syntax was clarified regarding the `all` action. If `all` is specified, others actions are not output by the `show` command.

(l) The conveniences variables names all have an underscore prepended to them. Also some are described to be read-only and others not. The `_p` variable is described to work syntactically and explictly states that the value is an empty string if `_n` has no parent.

(m) The arguments to the `server enable` and `server disable` commands is a list of server execution object. Server configuration object names are not allowed.

(n) The value of the `-comp` option to the `log list` command is now a list. Also the output of this command no longer prints the message number, just the message text.

(o) The end of the description of the `show` operation in the *Common Operations* section is more accurate with respect to the `registry show` command an ERA's.

(p) The description of principal names in the `principal` object was made more precise.

(q) The example in `principal create` was corrected so that `-alias` has a value of `yes`.

(r) The annotation field of an ERA is now defined to be limited to PCS.

(s) The encoding `confidential` has been removed from the `schema` object.

(t) The `schema create` command does not allow the `uuid` attribute to be specified if the argument is a list of names.

(u) It was specified that the `schema delete` command removes all ERA instances of the schema entry.

(v) The `schema modify` command can only modify modifiable attributes.

(w) The `log modify` command only takes a `-change` option.

(x) A `schema catalog` operation was added.

(y) The `CDS_ObjectUUID` on the `directory` object is read-only to the user and is set by the system at creation time.

(z) The `lastsync` attribute is defined to be the time of the start of the last attempted synchronization.

(aa) It is now stated explicitly that the child object in a `directory add` operation must exist or the command will return an error.

(ab) The `dts` object can now affect a `dtsd` on another host via an argument to a `dts` command.

(ac) Added a new `secval` object.

(ad) Added a `-single` option to `object create` and and `directory create`. Added `-single` and `-types` to `object modify` and `directory modify`.

(ae) The `login` command explicitly states that it updates the `_u` and `_c` convenience variables.

(af) Routing specifiers are now called serviceability routing specifiers. A string syntax was added to both serviceablity and debug routing specifiers. A new field was added to both the tcl and string syntax for private data. The Tcl syntax of Debug Routing Specifiers was changed so that the second element is a list instead of a dot-separated string.

(ag) A `-schema` option was added to the `object show` and `directory show` commands to return whether each attribute is single or multi-valued.

(ah) Removed the `-recursive` option from `rpcgroup delete` and `rpcprofile delete` and from the `list` operation of both of those objects.

(ai) The specification of an argument and of the relevant `_s()` convenience variable was made clear for the `registry`, `dts`, and `aud` commands. Each can take an argument of a single server name or string binding, not a list. The argument takes precedence over the relevant `_s()` convenience variable if set.

(aj) The `server modify` command no longer accepts an `-executing` option. The `server start` command now returns the UUID of the started server on success.

(ak) A `-binary` option was added to the `create`, `modify` and `show` operations of the `server` object. The description of the `data` attribute was updated.

(al) The `create` operation on the `server`, `hostdata` and `keytab` objects now accepts either an `-attributes` option or attribute options as other commands do.

(am) The `repository` flag has been removed from the `flags` attribute of the `server` object.

(an) The `catalog` operation on the `server`, `hostdata` and `keytab` objects now takes a `-simplename` option.

(ao) The syntax of the `executing` attribute of the `server` object was defined (this was formerly called `srvrexec`).

(ap) Completely removed the `-recursive` option.

(aq) Specified the order of precedence for `_s(cds)` and `-clearinghouse` in the `directory` object.

(ar) The `-force` and `-only` options to the `registry delete` command may not be used together.

(as) The options on the `registry show` command have been changed slightly. The old `-replica` with no value is now `-replica`. Also `-master` replaces the old `-replica` option with a value.

(at) Updated the description of the `endpoint` object. Removed the `-host` option. The `endpoint` may only affect a local endpoint map. The `server` object using the `dced` API's can perform some remotes operations on `dced` (not `rpcd`) endpoint maps. Also clarified behavior when lists are passed in as values.

(au) The argument to `principal`, `group`, `organization`, and `account` objects is a list of names, but if any are not fully qualified, then none may be. Non-fully qualified names refer to the name in the cell identified by `_s(sec)` if set or to the default cell of the local host if not.

(av) Added the `uuid` and `name` objects. Removed the `expand` miscellaneous command as this is now `name expand`.

(aw) Added a `utc` object to convert and perform arithmetic on UTC timestamps.

(ax) The values of the `starton` attribute of the `server` object have been changed to bring them in line with [RFC 47.3]. Also added attributes: `uid`, `gid`, `directory`, and `principals`.

(ay) Updated Apendix A.

(az) You must specify either `-registry` or `-version` or both on any `keytab add` command.

(ba) Added the `aud rewind` command.

(bb) The `registry delete` and `registry synchronixe` commands must either have an argument or `_s(sec)` must be set.

(bc) The `clock set` command now takes the time as the value of a required `-to` option and the argument can be a remote `dtsd` to contact.

(bd) Added the `attrlist` object.

(be) The argument to the `registry` commands and its handling of `_s(sec)` has been further specified.

(bf) Specified that a UUID may be returned in an ACL entry if `dcecp` cannot convert from the UUID stored in an ACL to a name.

(bg) Added a new `cellalias` object for CDS Hierarchica cells support.

(bh) Specified that blank lines are given between multiple objects displayed at once, e.g., in `show` commands.

(bi) Added description of sequence numbers in `registry` object.

(bj) Five new states were added to the `status` attribute of the `registry` object.

(bk) Added a `registry dump` command.

(bl) Added `name` and `type` attributes to the info displayed by `registry show -master`.

(bm) Added `registry verify`.

(bn) Added `-randompwd` option to `account modify`.

(bo) Removed ERA's from the `registry` object.

(bp) Changed `-c` option to `dcecp` to take one value instead of the rest of the arguments. This is so adding more options to `dcecp` will not force them to be position dependent.

(bq) Added the `-certify` option to `login`.

(br) Specified the order of the output of `registry show`.

(bs) Specified `-noupdate` on the appropriate RPC commands

(bt) Changed the default value of the `courierrole` attribute from `noncourier` to `backup`.

(bu) Secified the `user`, `host`, and `cell` task scripts.

(bv) Specified the `-s` option to the `dcecp` program.

(bw) Added the global variable `dcecp_verbose_errors` which causes `dcecp` to output the entire contents of `errorInfo` to `stderr` in the event of an error.

(bx) Renamed the `schema` object `xattrschema` to match the name of the namespace objects and the ACL manager.

(by) Added the following commands:

```
dts catalog
clock compare
clearinghouse catalog
```

(bz) Added `name get`.

(ca) Added the `scope` attribute to `xattrschema`.

(cb) Corrected argument handling of `registry catalog`.

(cc) Renamed `cellalias` to `cdsalias`.

(cd) Changed `cdsalias set -child` to `cdsalias connect`.

(ce) Specified the output of `cdscache show`.

(cf) Added `cellalias` object for big hierachical cells script.

(cg) Addded the following commands:

```
clearinghouse checkpoint
clearinghouse repair
clearinghouse verify
```

(ch) Added that the `applydefs`, `intercell`, `scope`, and `unique` attributes on the `xattrschema` object are only advisory in DCE 1.1.

(ci) Changed the default value of the `tolerance` attribute of the `dts` object to 5 minutes as per DCE CR 11089.

(cj) The `user` commands take a list of user names as an argument.

(ck) Fixed typo in `object show -schema` description.

(cl) Added that the user must be `root` to run `cdscache dump`.

(cm) Clarified the descriptions of the `goodsince` and `server` attributes of the `account` object.

(cn) Changed the description of the `CDS_ObjectUUID` attribute in the `directory` and `clearinghouse` objects.

(co) Clarified descriptions of all the attributes of the `directory` object, particularly the `CDS_INCHName`, `CDS_UpgradeTo`, and `CDS_RingPointer` attributes.

(cp) Updated output of CDS commands. The `cdscache show` command output has been changed, also the following attributes of `directory` and `clearinghouse show` commands have changed:

```
Clearinghouse_uuid CH_UUID
Clearinghouse_name CH_Name
dir_uuid           Dir_UUID
directory          Dir_Name
```

(cq) Added `-local` to all `keytab` commands.

(cr) Typo in `masteruuid` attribute of `registry`, the value is of type UUID, not integer.

(cs) Modified description of `clock set` so that `-to` is optional.

(ct) The `rpcgroup add` command now allows the user to enter duplicates entries and does *not* flag an error.

(cu) Changed the format of ERA's of encoding type `void`.

(cv) Added the `registry connect` command.

(cw) There is a new optional argument `registry catalog` to specify a single name of a cell or a string binding.

(cx) Specified the syntax of multiple routing and debug routing specifiers.

(cy) Added the `_h` convenience variable.

(cz) Added the `registry set` command.

(da) Specified that `import` only reports an error if no match was found in *any* of the names specified in the argument list.

(db) Renamed `name` in the `service` attribute to be `entryname`.

(dc) Removed the `-randompwd` option from the `account modify` command and added the `account generate` command. This is to avoid the case where a password can be changed in the registry but the server crashes before the user is informed of the new password.

(dd) Added the `_b(sec)` convenience variable to replace portions of `_s(sec)`. This way `_s` is only set by the user not by `dcecp`, and `_b` is only set by `dcecp` and not by the user. Much less confusion can reign.

(de) Added the `operations` attribute to the `server` object. It is only available for input, it is not output.

(df) Specified that the argument to `registry show -master` can be a cellname.

(dg) Added the `-noprivacy` option to all `keytab` commands.

(dh) Added a new section on *Warning Messages*.

(di) `clearinghouse checkpoint` is now `clearinghouse initiate`.

(dj) Cleaned up the new `clearinghouse` commands, particularly `verify`.

(dk) Added `secval status`.

(dl) The `acl show` command describes the causes of UUID's being displayed instead of names more completely.

(dm) Added a `-xattrs` option to `principal`, `group`, `organization` and `registry show` operations. Some also get a `-all`.

(dn) Added `GOESTO` routing specifier.

(do) Added the `-uuid` option to `endpoint show`.

(dp) Added the `-simplename` option to `cell show` and included some examples.

(dq) Changed the `stop` value of the `aud stostrategy` attribute to `save`.

(dr) `endpoint show` and `delete` take an optional argument of a single string binding to identify an endpoint map to contact.

(ds) Added optional `-object` option to `resolve` command.

(dt) Provided more argument details for the `clock` object.

(du) Removed the argument to `cdsalias connect`.

1.2. Changes For Future Versions

This section lists some ideas for `dcecp` which will not appear in the DCE 1.1 release but which should be considered for future releases. Several ideas for future support have been thought out, but since they are not appearing in DCE 1.1 they should not be included in this specification. They are included here so that the ideas can be considered when the support of this functionality is required.

(a) If the NSI ever supports more than the `dce` syntax, a `-syntax` option should be added to the relevant `rpcentry`, `rpcgroup`, and `rpcprofile` commands. The NSI API's take a syntax argument, but since dce is the only supported name syntax, there is no current need to export this functionality to the user. The value for the dce syntax should be `dce`.

(b) A `search` operation could be added to several objects so that filtering could be done for searches. By default names of matching objects should be returned so that can be used as input into other commands. An option could be used to specify that the contents of the objects should be returned. The argument, when possible, should optional so that all instances of the object are searched (see the `catalog` operation). If the argument is present it limits the searches scope. Different options can be supported for different types of search criteria. The following come to mind: `-regexp` for regular expression matching, `-match` for globing style matching, `-equal` for string equality, `-filter` for complex filters (such as X.500 filters). Not all options need be supported for each object, but these options should be used for the described functionality when present.

(c) It should be possible for `dcecp` to support background tasks. If an `&` is at the end of the line, a thread should be spawned to perform the operation. Users would probably want some form of job control and that could get complicated.

(d) A new convenience variable `_a` should be created to allow specification of the authentication and protection level used by RPC's initiated by `dcecp`. There are conflicting goals of providing full access to end users and providing a simple interface to complicated concepts available in DCE. Current thoughts include:

(i) The variable should be a Tcl array as `_s` is, allowing fine grain settings for each component. Should there be a general default as well?

(ii) The values should be a list of two elements specifying the authentication service and the protection level. The actual values should be taken from the last word in the constants defined for the `rpc_binding_set_auth_info()` call. It could also be a list of such values allowing an order choice in the event of failure. Or it could somehow try to collapse the two concepts into one simple string which would hopefully be easy and adequate for the user (e.g., `unauth`, `auth`, `private`).

(iii) If the specified level cannot be used (e.g., the user is not authenticated), the call should fail. Or if a list of values is used, then it could be an ordered list to try in the event of failure.

(iv) The effect of the addition of public key technology to DCE on this variable is not clearly understood. Would a ever want to specify secret vs public? Would they ever have the choice or would a login mechanism make that choice for them?

(e) Several of the `host` task scripts should be extended to allow remote operations where possible, particularlly `configure` and `unconfigure`. This may require some new restrictions (e.g., `dced` must be `running`), or some new options to the commands.

(f) A new `_s(sch)` should be defined for schema entries. Perhaps a corresponding `_b(sch)` as well.

2. TERMINOLOGY

The following are the terms used in this document.

(a) **Administrator**

The person (or persons) responsible for the installation and maintenance of a DCE Cell. They perform various administrative operations and **tasks**. While performing these duties they must have the proper permissions, so they must be members of the appropriate administrative groups.

(b) **Argument**

A parameter given to a command. **Commands** take zero or more arguments followed by zero or more **options**.

(c) **Command**

Used in this document to mean a Tcl command or `dcecp` script that will be executed if entered as a command (Tcl provides a $PATH-like facility). Almost all `dcecp` commands are named for the type of **object** that they operate on (e.g., ''the `acl` command''). This word is also used to refer to object-operation pairs (e.g., ''the `acl show` command'').

(d) **Control Programs**

All of the administrative user interfaces provided with DCE 1.0. This includes but is not limited to: `rpccp`, `cdscp`, `rgy_edit`, `acl_edit`, and `dtscp`.

(e) **Element**

Existing DCE terminology uses this term to refer to elements of RPC profiles. The `dcecp` uses **member** for this purpose. In `dcecp` (and in Tcl), element is used to refer to the contents of lists.

(f) **ERA**

ERA stands for Extended Registry Attribute. These are extensions to the DCE 1.0 security service added in DCE 1.1 that allow arbitrary attributes to be added to objects in the registry (i.e., principals, groups, organization, and policy objects). They are defined by a schema which is stored in the registry as well.

(g) **Member**

Existing DCE terminology uses this term to refer solely to members of groups (both RPC groups and registry groups). Other terms are used to refer to objects in other types of containers (e.g., RPC profiles contain **elements**). The arbitrary choice of words has caused confusion for some users. The `dcecp` uses the word `member` to refer to anything stored in container.

(h) **Operation**

The commands in `dcecp` are usually named for the type of object they act on. The first argument to the command is usually the operation to perform on that object.

(i) **Option**

Tagged arguments to commands. All options begin with a `-` (dash) and are found after all **arguments** in `dcecp` command lines. Some options are optional and some are required.[2] Some options take values and some do not.

(j) **String Syntax**

A format of a data structure that is understood by `dcecp` and existing control programs. In some cases, data structures have two syntaxes, one that is used in existing control programs, and one that is more natural for `dcecp`. Both are supported by `dcecp`.

(k) **Task**

A high level operation that an administrator performs, such as add a new user, or configure a new machine. A task is above the level of DCE components (e.g, add principal and account to registry), and usually involves operations on several different components. Usually implemented as dcecp task scripts.

(l) **Tcl**

The embeddable Tool Command Language. Tcl refers to both the language and to a library (`libtcl.a`) that can be linked into a program which provides it. Commands can be added to the language via both C code and Tcl scripts. The `dcecp` language can be considered a dialect of Tcl, i.e., it is Tcl plus some DCE-specific commands.

(m) **Tcl Syntax**

A format of a data structure that is understood by `dcecp`. In some cases, data structures have two syntaxes, one that is used in existing control programs, and one that is more natural for `dcecp`. Both are supported by `dcecp`.

2.1. Abbreviations

The following are abbreviations used in object names, attribute names, and option names:

```
account         acct
attribute       attr
clearinghouse   ch
database        db
default         def
expiration      exp
```

2. This may seem strange, but it follows OSF conventions.

```
length          len
lifetime        life
message         msg
observed        obs
password        pwd
project         proj
standard        std
synchronization sync
ticket          tkt
update          upd
```

3. TARGET

The `dcecp` and is meant to be used by DCE administrators. The emphasis of the first release will be on a commonly performed tasks of DCE cell administrators. It will have a command-line interface only.

Developers will probably find an advantage to using `dcecp`, however they are not the primary target user.[3] For example, the most commonly performed task of a DCE developer is to configure a new cell. However, it is not expected that this will be the same for DCE administrators.

4. GOALS AND NON-GOALS

The following are goals for `dcecp`:

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

(b) 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.

(c) The `dcecp` 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. See Appendix A for a list of existing control programs commands and the `dcecp` commands that replace them.

(d) Any new administrative interfaces required by new DCE 1.1 functionality should be provided via `dcecp` commands rather than new separate control programs.

(e) Commands should 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.

(f) The `dcecp` must be portable to a variety of different platforms and management situations. Vendors must be able to integrate it into their own management systems.

The following are not goals of `dcecp`:

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

(b) 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 `dcecp`.

---

3. It is expected that during the course of DCE 1.1 development, developers who have access to early versions of `dcecp`, will produce task-oriented scripts that may be of use to DCE administrators. As many of these as is possible and practical will be included in the first release of `dcecp`.

(c) This is not DME. The `dcecp` is not meant to replace the need for DME, but rather to substitute for its absence in a very limited manner (but better than the existing control programs have done). Nothing in the design of the `dcecp` should preclude the use of DME in the future.

5. REQUIREMENTS

The following are the requirements of `dcecp` as specified in [RFC30] and how they are met by the program described in this functional specification:

(a) Existing Control Program Functionality

- (i) Stop servers — Met with `server stop` operation.
- (ii) Show/add/modify/delete data — Met with the `show`, `create`, `modify`, `delete`, `add`, and `remove` operations on various commands.
- (iii) Show status — Met with various `show` operations
- (iv) Remote operations — Met by most commands, see the *User Interface* section for details.

(b) Usability

- (i) Common syntax for all functions — Met with Tcl and designed syntax.
- (ii) Help — Met with `help` operations on all commands.
- (iii) Provide command line recall/editing — Met with command-line editing facility.
- (iv) Scriptability — Met with Tcl's `source` command and with the interpretation of an argument as a script file to execute.

(c) New Functionality

- (i) Administer serviceability — Met with the `log` command.
- (ii) Administer auditing — Met with new commands: `aud`, `audevents`, `audfilter`, and `audtrail`.
- (iii) Programmability — Met with use of Tcl.
- (iv) Extensibility — Met with Tcl's `proc` command.
- (v) Bind to different servers or data — Met with separate commands, the `s` convenience variable, the `-clearinghouse`, etc.
- (vi) Shell escape — Met with the `shell` command and Tcl's `exec` command.
- (vii) Administration of little files — Met with `hostdata` command.
- (viii) DCE ping — Met with the `ping` operation.
- (ix) Security — Will be done by existing ACL's.

(d) Meet Other DCE 1.1 Goals

- (i) Code Cleanup — Will follow the DCE Coding Style Guide [RFC 34].
- (ii) Internationalization — The new `dcecp` code will follow proper internationalization coding practices. Tcl will be internationalized.
- (iii) Serviceability — Will use `sams` for message catalogs and messaging API's. The three letter serviceability component name of `dcecp` will be `dcp`. The routine `dce_sprintf` will be used for all messages. See [RFC 23], [RFC 24], and [RFC 34] for details.

6. FUNCTIONAL DEFINITION

Many people seem to get confused trying to understand the relationship between `dcecp` and Tcl. It is a close relationship, but understanding which commands are Tcl commands and which are DCE commands is unimportant.

A user of `dcecp` starts the control program and enters interactive commands at the `dcecp>` prompt. There are other use models (e.g., writing scripts and entering commands from the shell command line) but they are less common. An important usability feature of `dcecp` is that it provides command-line recall and editing similar to `ksh`'s Emacs mode. At the prompt the user can enter base Tcl commands such as:

```
dcecp> echo foo
foo
dcecp> lsort {a c e b d}
```

```
a b c d e
dcecp> foreach i [lsort {a c e b d}] {puts stdout $i}
a
b
c
d
e
```

The Tcl library implements an interpreter that understands all the base Tcl commands. See Appendix C for a list of Tcl commands. Any program using this library has access to these same commands.[4] A typical Tcl program usually implements new project-specific commands. Not surprisingly, `dcecp` implements commands that manipulate DCE data. Users can at the same `dcecp>` prompt enter DCE commands such as:

```
dcecp> object delete /.:/hosts/absolut
dcecp> acl modify /.:/hosts -add {user melman -rwx}
dcecp> principal create melman -fullname {Howard Melman}
```

To the user of `dcecp` there is no visible difference between a base Tcl command and a command that manipulates DCE data. They might see the base Tcl commands when running other programs that use Tcl, but they will only see the DCE commands in `dcecp`. The Tcl and DCE commands can be combined (output not shown for space reasons):

```
dcecp> foreach i [dir list /.: -directories] {dir show $i}
```

This allows an advanced user to write extensible scripts, a feature that we deem very important. For a list of base Tcl commands, see [TCL]. See the *User Interfaces* section for all other details about the usage of `dcecp`.

7. DATA STRUCTURES

This section provides a brief introduction to Tcl data types and then presents several `dcecp` data structures. For a complete reference of Tcl data types, see [TCL MAN].

In Tcl, there is only one data type, strings. All commands, expressions, variable values, and procedure return values are strings. Some Tcl commands interpret the strings in different manners. For example, `eval` treats its **argument** as an expression. The argument it receives is a string. If the argument is `4 + 5` then the returned result is the string `9`.

Other Tcl commands interpret their string arguments as lists. In Tcl lists consist of elements separated by white space (space, tab, or newline). The following are three examples of lists with three elements:

```
tom dick harry

a b c

1 2 !
```

There are three meta-characters that are treated specially when parsing lists. They are:

(a) Braces are used for grouping. If a left brace is encountered, then the element is terminated by the matching right brace, instead of the next whitespace. Braces nest. The following are also examples of lists with three elements:

4. This is orthogonal to DCE. The only part of DCE that will use `libtcl` is `dcecp`. The Tcl library will *not* be part of `libdce`. The goal of DCE is to ship `dcecp` not Tcl, as such if we will not install `libtcl` and the headers for general availability. This does not prevent vendor's from doing this. IT affects only the reference implementation of `dce_config`. The AES mentions neither `dcecp` nor Tcl. Tcl is freely available to all from the internet.

```
tom {dick harry} {sue mary jane}

a {b {c d e}} f
```

(b) Double quotes are also used for grouping, however, there are two differences between them and braces. First they don't nest. Second Tcl will perform command, variable, and backslash substitutions for items within double quotes. Tcl does not perform these substitutions when braces are used.

(c) Backslashes are used to escape meta-characters, and to insert non-printing characters. Standard substitutions occur such as `\t` is a tab character, and `\n` is a newline. Braces and double quotes can be inserted in lists with backslashes as follows, again these example lists all have three elements:

```
a b\ c d

a b \}
```

The net effect of using these meta-characters and Tcl commands on a command line is that lists appear as elements surrounded by braces, and the top level of braces is not present on output. This last point is tricky; it has to do with how commands are parsed. The best way to explain it is with an example. The Tcl command `lindex` takes two arguments, a list and an integer (remember, both of these are just strings and it is the `lindex` command that interprets these string arguments specially). It returns the element from the list indicated by the second argument. Elements in lists are numbered from zero. So the following command:

```
lindex {a b c} 1
```

returns the string `b`. The braces in the example are used to group the elements of the list so that they are passed as one argument to the lindex command. Now, the following example returns a list:

```
lindex {a {b c} d} 1
```

The list returned is `b c`, there are no braces around it, but it is a perfectly valid list. To prove this, pass it to the `llength` command which takes one argument, a list, and returns the length as an integer (in Tcl, square brackets perform command substitution, much like backquotes do in command shells):

```
llength [lindex {a {b c} d} 1]
```

This returns the value `2` since the list `b c` has two elements.

In `dcecp`, there are several data structures used to represent DCE objects such as attributes, bindings, and ACL's. Most of these had some representation in the existing control programs. The `dcecp` often supports these representations for backward compatibility, as they will be recognized by the administrator. This is known as the **string syntax** of the data structure. Often `dcecp` provides a representation that is more easily handled by Tcl for these data structures, usually they are lists. This representation is known as the **Tcl syntax** of the data structure.

7.1. Attributes

In `dcecp`, attributes are represented as a list. The first element is the attribute type. This is usually some string name, but in some cases, OID's or UUID's are acceptable. The following elements in the list are the values. If the attribute is single-valued then the attribute list will have two elements, multi-valued attributes will have more. Attribute lists are lists of attributes. The following is an attribute list of two single-valued attributes:

```
{acctlife 2} {pwdlife none}
```

7.2. ACL's

An ACL is an Access Control List which consists of a set of ACL Entries. Each ACL Entry is defined as a type, an optional key or value, and a required set of permissions. In access checks, the permissions granted to a user are established by checking the type and key of each ACL Entry and or'ing the permissions described therein.

Each application is free to define new permission bits. The defined types are listed below:[5]

(a) `user_obj`
(b) `group_obj`
(c) `other_obj`
(d) `user`
(e) `group`
(f) `foreign_user`
(g) `foreign_group`
(h) `foreign_other`
(i) `any_other`
(j) `mask_obj`
(k) `unauthenticated`
(l) `extended`
(m) `user_obj_delegate`
(n) `group_obj_delegate`
(o) `other_delegate`
(p) `user_delegate`
(q) `group_delegate`
(r) `foreign_user_delegate`
(s) `foreign_group_delegate`
(t) `foreign_other_delegate`
(u) `any_other_delegate`

All ACL type names are case insensitive. The following are all legal: `user`, `USER`, and `uSer`. The value of the key may be either a name (e.g., a principal or group name) or a UUID. On output, `dcecp` will always try to generate a name from the UUID stored in the ACL, but if this is not possible, a UUID will be returned.

7.2.1. String syntax

The string syntax of ACL Entries is as follows:

type`[:`*key*`]:`*permissions*

The permission bits need not include `-`'s for permissions not granted and may be given in any order. Examples of nine ACL Entries follow (the last appears on two lines for space reasons[6]):

```
unauthenticated:-r-----
user_obj:crwx---
user:britten:crwx---
user:mahler:-rwx---
foreign_user:/.../C=US/O=OSF/OU=dce/pro/bach:crwxidt
group_obj:-rwx---
group:dds:-rwx---
any_other:-r-----
extended:c417faf8-8340-11c9-ace3-\
08001e5559bb.a.b.c.a1.4.0a0b0c0d:-rwx---
```

7.2.2. Tcl syntax

The core Tcl Syntax of ACL Entries is a list of two or three elements. The first element is the type, the optional second element is the key, and the last element is the set of permission bits. The permission bits are represented as a single character if the permission is granted and as a `-` (dash) if it is not. An ACL is a list of ACL Entries. An example of an ACL is as follows:

---

5. The new ''delegate'' types are to be added during DCE 1.1 development to support delegation.

6. Line-terminating ''\'' denotes line-folding of long lines, sometimes as a display meta-character and sometimes as a Tcl meta-character (the distinction will always be clear from context).

```
{unauthenticated -r-----}
{user_obj crwx---}
{user britten crwx---}
{user mahler -rwx---}
{foreign_user /.../C=US/O=OSF/OU=dce/pro/bach crwxidt}
{group_obj -rwx---}
{group dds -rwx---}
{any_other -r-----},
{extended c417faf8-8340-11c9-ace3-\
08001e5559bb.a.b.c.a1.4.0a0b0c0d -rwx---}
```

On output the above Tcl Syntax will be used, with one addition. If due to masking[7] there are ineffective bits in an ACL Entry, the entry will be extended with two elements. The first is the string `effective` and the second is the set of effective permissions. This will only be added for those ACL Entries that have ineffective bits. For example:

```
{mask_obj -r-----}
{user_obj crwx---}
{user britten crwx--- effective -r-----}
```

On input, these two elements are ignored. Also on input, the permission bits need not include `-`'s for permissions not granted and may be given in any order. For example, if the user wanted to enter the above ACL to a command, all that they need to type is:

```
{mask_obj r}
{user_obj crwx}
{user britten wcrx}
```

7.3. Bindings

In RPC, all bindings are represented as string bindings which contain four pieces of information: a protocol sequence, a network address, an endpoint, and an optional object UUID.

7.3.1. String syntax

The string syntax of a string binding is as follows:

object-UUID`@`*protseq*`:`*address*`[`*endpoint*`]`

The *endpoint* can be preceded by the string `endpoint=`. There can be no whitespace in the string binding. Examples follow:

```
ncadg_udp_ip:130.105.96.3[1234]

001c8234-74ff-1be9-b11c-08002b0f59bb@ncadg_udp_ip:\
130.105.96.3[1234]
```

Note that in Tcl, square brackets are used to indicate command substitution. So, if a string binding in the string syntax is entered, the brackets must be preceded by a backslash (i.e., escaped) or enclosed in braces (i.e., quoted). Therefore, the above example would not always be accepted by the Tcl parser, two corrected examples of the first string binding follow:

```
{ncadg_udp_ip:130.105.96.3[1234]}

ncadg_udp_ip:130.105.96.3\[1234\]
```

---

7. See [POSIX.6] and [USERGD] for details on masks in ACL's.

7.3.2. Tcl syntax

The Tcl Syntax is a list of 3 or 4 elements as follows:

protseq address endpoint

or

object-UUID protseq address

or

object-UUID protseq address endpoint

Examples follow:

```
ncadg_udp_ip 130.105.96.3 1234

001c8234-74ff-1be9-b11c-08002b0f59bb \
ncadg_udp_ip 130.105.96.3 1234

001c8234-74ff-1be9-b11c-08002b0f59bb \
ncadg_udp_ip 130.105.96.3
```

7.4. Interface Identifiers

An interface identifier describes an interface. Specifically, it contains the UUID, major version number, and minor version number of an interface.

7.4.1. String syntax

The string syntax is as follows, where *UUID* is the string syntax of a UUID.

UUID,*major*.*minor*

An example follows:

```
0067b39e-67ac-1be9-abb5-08002b0f59bb,1.2
```

7.4.2. Tcl syntax

The Tcl Syntax is a list of two elements: the UUID in its string syntax and the major and minor version numbers separated by period. An example follows:

```
0067b39e-67ac-1be9-abb5-08002b0f59bb 1.2
```

7.5. Serviceability Routing Specifiers

Serviceability routing information [RFC 24] is specified as a list of serviceability routing elements. Each routing element contains four fields containing PCS data:

(a) The first field specifies a message severity. It must be one of the following:

- (i) `FATAL` Fatal error, about to exit.
- (ii) `EXIT` Normal exit.
- (iii) `WARNING` Error detected, program proceeding.
- (iv) `NOTICE` Informational notice.
- (v) `VERBOSE` Verbose informational notice.

(b) The second field specifies how the messages should be processed. It must be one of the following:

- (i) `GOESTO` Set routing the same as another severity field.
- (ii) `BINFILE` Write binary log entry.

(iii) `TEXTFILE` Write human-readable text.

(iv) `FILE` Equivalent to `TEXTFILE`.

(v) `DISCARD` Do not record this level.

(vi) `STDOUT` Write human-readable text to standard output.

(vii) `STDERR` Write human-readable text to standard error.

Note that any debug messages (calls to `DCE_SVC_DEBUG`) that are routed through a `BINFILE` route will lose the types of any arguments.

(c) The third field specifies where the messages should be sent. Presently, this will always be a filename, although it can be blank for the last three specifiers given above. This field may contain a `%ld` string which will be replaced by the process-ID of the currently-running process. Filenames may *not* contain colons, semi-colons or spaces. For the `GOESTO` sepcifier, the where field contains the name of another severity field. E.g., you can set notices to be the same as errors, whatever they are set to. It is a reference not a copy.

A limited form of log-rolling is available. The where field,, may be followed by `.`*`gens`*`.`*`count`* where *`gens`* is an integer less then 100 indicating the number of files to be kept and *`count`* is an integer indicating the number of entries per file. No more then *`count`* entries will be written to a file, before a new file is started. After *`gens`* files have been written, the files wrap back around to the beginning. The file names are formed by appending `.`*`gen`* to the end of the filename, where *`gen`* runs from one to the last generation number.

(d) The optional fourth field is used for application specific information.

Note that the second, third, and optionally the fourth field may be duplicated to allow for multiple routing specifiers. See the specific syntaxes below for example of how this can be specified.

7.5.1. String Syntax

The string syntax is as follows:

```
severity:how:where[:private]
```

For example, to send fatal errors to the console, send errors to a log-rolled file, ignore normal exit reports, send everything to a log file, send informational messages to a temporary binary log, and send warnings to stdout with private data, use the following serviceability routing specification:

```
FATAL:TEXTFILE:/dev/console
ERROR:TEXTFILE:/tmp/errors.5.100
EXIT:DISCARD:
*:FILE:/tmp/svc-log
NOTICE:BINFILE:/tmp/log%ld:app_private_data
WARNING:STDOUT::app_private_data
```

Multiple routings can be specified by separating several *`how`*, *`where`* and optionally *`private`* fields with a semi-colon. E.g.:

```
{ERROR:STDERR:;STDOUT:;FILE:/tmp/jas}
{FATAL:FILE:/tmp/jas:private;STDERR:}
```

7.5.2. Tcl Syntax

The Tcl syntax of a serviceability routing specification is a list of four elements. The first element is the severity, the second is the how specification, the third is the where specification, and the fourth is the private field

```
{severity how where private}
```

The above example in Tcl syntax is as follows:

```
{{FATAL TEXTFILE /dev/console}
 {ERROR TEXTFILE /tmp/errors.5.100}
 {EXIT DISCARD}
 {* FILE /tmp/svc-log}
 {NOTICE BINFILE /tmp/log%ld app_private_data}
 {WARNING STDOUT {} app_private_data}}
```

Multiple routings can be specified by using a Tcl list for the *how*, *where* and optionally *private* fields. E.g.:

```
{ERROR {STDERR STDOUT {FILE /tmp/jas}}}
{FATAL {{FILE /tmp/jas private} STDERR}}
```

7.6. Debug Routing Specifiers

A debug routing specification is a list of debug routing elements. Each debug routing element contains four fields with an optional fifth field:

(a) The first field is the name of a serviceability component.

(b) The second field is a list of period separated subcomponents and levels. If the subcomponent is an asterisk then the level is set for all subcomponents of that component. The level should be a number between zero and nine. This list is parsed in order, so the asterisk may be used to set the initial level for all subcomponents; later entries override earlier ones.

(c) The third field specifies how the messages should be processed. It is identical to the second field of a serviceability routing element.

(d) The fourth field specifies where the messages should be sent. It is identical to the third field of a serviceability routing element.

(e) The fifth field is optional and can be used to specify application specific data.

Note that the third, fourth, and optionally the fifth field may be duplicated to allow for multiple debug routing specifiers. See the specific syntaxes below for example of how this can be specified.

7.6.1. String Syntax

The string syntax is as follows:

```
component:sub-comp.level:how:where[:private]
```

Multiple sub-component and level specifications can be entered separated by commas. For example:

```
dts:*.1:TEXTFILE:/dev/console
dts:msgs.4:DISCARD:
dts:msgs.4,states.6:DISCARD:
cds:clerk.9:BINFILE:/tmp/log%ld:app_private_data
cds:general.1:STDOUT::app_private_data
```

Multiple routings can be specified by separating several *how*, *where* and optionally *private* fields with a semi-colon. E.g.:

```
{cds:general.7:FILE:/tmp/jas;STDOUT:;STDERR:}
{dts:msgs.2,general.7:STDERR:;STDOUT:;FILE:/tmp/jas}}}
```

7.6.2. Tcl Syntax

The Tcl syntax of a debug routing specification is a list of four or five elements, one for each of the above fields. The second element is a list and may be a list of lists. For example, to set RPC debugging at a minimal level so that all messages go to standard error, use the following:

```
{rpc {* 1} STDERR}
```

The following example sets DTS debugging messages for the general, events, and msgs subcomponents (at the levels indicated for each) to go to a logfile:

```
{dts {{general 1} {events 0} {msgs 3}} TEXTFILE /tmp/log}
```

Multiple sub-component and level specifications can be entered as a tcl list. The Tcl syntax of the example in the string syntax section is as follows:

```
{dts {* 1} TEXTFILE /dev/console}
{dts {msgs 4} DISCARD}
{dts {{msgs 4} {states 6}} DISCARD}
{cds {clerk 9} BINFILE /tmp/log%ld app_private_data}
{cds {general 1} STDOUT  app_private_data}
```

Multiple routings can be specified by using a Tcl list for the *how*, *where* and optionally *private* fields. E.g.:

```
{cds {general 7} {{FILE /tmp/jas} STDOUT STDERR}}
{dts {{msgs 2} {general 7}} {STDERR STDOUT {FILE /tmp/jas}}}
```

7.7. Other Data Types

There are three other data types mentioned in this document: UUID's, UTC times, and Names. They have no Tcl Syntax. For a description of these string syntaxes, see the DCE Documentation.

8. USER INTERFACES

The `dcecp` will present a command-line interface to the user with command-line recall and editing capabilities. The commands and command-line editing interfaces are described below.

This section describes the following in various sub-sections:

(a) *Invocation* — How `dcecp` can be started and its modes of operation.

(b) *Initialization* — The various scripts that are read when `dcecp` is started.

(c) *Command Syntax* — A description of the design of the command syntax. Also how to invoke external commands, and how `dcecp` deals with commands it does not know.

(d) *Abbreviations* — The `dcecp` allows objects, operations, and options to be abbreviated to the shortest unique string.

(e) *Convenience Variables* — Describes builtin variables to make interactive use easier.

(f) *Error Handling* — The overall approach to return values and error handling.

(g) *Help System* — Describes the available on-line help.

(h) *Common Operations* — These are operations that are common to many but not all objects.

(i) *Common Options* — These are options that are implemented in more than one object.

(j) *Objects* — These are the object commands that `dcecp` will support. Each sub-section describes one object, including the operations that it supports. Variations in the operations of a particular object versus the common operations are described in this section, as well as any non-common operations supported by the object.

(k) *Tasks* — These are the administrative tasks that `dcecp` will support. They are implemented as object commands so as to have similar syntaxes as other `dcecp` commands. They are more complex and are typically implemented in Tcl code.

(l) *Command-line Editing* — A description of the command-line editing interface.

8.1. Invocation

The `dcecp` is a command-line user interface for administrative commands. There are several methods of invocation:

(a) The user starts `dcecp` and then sees the `dcecp` prompt:

```
dcecp>
```

The above is the default prompt. This can be changed using the standard Tcl `tcl_prompt1` and `tcl_prompt2` mechanisms. See [TCL MAN] for details.

(b) The user invokes `dcecp` with one argument which is a filename of a `dcecp` script. The script is run and then `dcecp` exits. This method of invocation allows interpreter files (those with `#!/bin/dcecp` as the first line) to work.

Tcl sets the `argc`, `argv`, and `arg0` variables to provide access to command-line arguments.

(c) The user invokes `dcecp` with the `-c` option followed by a value which is a set of commands (all on one line separated by `;`'s which must be quoted in the shell). The commands are executed and then `dcecp` terminates. For example:

```
$ dcecp -c "directory create /.:/foo"
```

The `dcecp` program also accepts a `-s` option which causes it not to contact any servers during initialization.. Specifically it will not try to inherit a login context from the invoking process. This can be useful when trying to invoke `dcecp` when DCE is not functioning properly.

8.2. Initialization

When `dcecp` is invoked, it executes the following scripts in the order shown. Note the syntax used for the script locations is Tcl.

(a) `[info library]/init.tcl`

Contains standard Tcl initialization commands on a per-host basis. Contains definitions for the `unknown` command and the `auto_load` facility.

(b) `$dcecp_library/init.dcecp`

Contains `dcecp`-specific startup information. The `dcecp` scripts implementing commands and tasks are stored in the directory identified by the Tcl variable `$dcecp_library`. The reference implementation will set `dcecp_library` to *dceshared*`/dcecp` by default.

(c) `$env(HOME)/.dcecprc`

This file stores user customizations.

8.3. Command Syntax

The `dcecp` commands are implemented as procedures in a Tcl interpreter. Commands return strings, and the main interpreter loop displays the results to the user. Errors are displayed as well. In most cases a Tcl object-oriented approach has been used to define the commands. In this approach commands represent a type of object, and have the same name as the object.

object operation name `[-`*option* `[`*value*`]]...`

The first argument to the command specifies an operation to perform on an object, the second argument identifies the particular object to operate on (usually by name) and the third and following arguments are options and values. Some commands might require one or more option. Options have a leading `-` and are a full word, not a single letter, but may be abbreviated (see below for details). Some options take a value which immediately follows the option. All arguments are strings, and all commands return strings as values. Some examples of Tcl commands are:

(a) `file exists` *filename*
(b) `history add` *command*
(c) `string compare` *string1 string2*
(d) `string toupper` *string*

The `dcecp` will define commands in a similar manner. Commands will be named for a type of object to operate on, and the operations will be implemented for each object. Operation names will be consistent whenever appropriate (e.g., `create` will always be used to create a new instance of the object, `delete` will always be used to delete an instance). This syntax will be used whenever possible, but in some cases (such as `quit`, `exit` and `shell`) such a mapping would be too contrived and will be avoided. The syntax is described below.

The commands available to the user are those known by the Tcl interpreter. In the `dcecp`, the interpreter will know about the Tcl built-in commands and the additional `dcecp` commands described below. If a command name that Tcl does not know about is invoked, then Tcl calls the Tcl built-in command `unknown` with the entered command and arguments. The user is free to define this command as they wish to have a variety of things happen. By default, Tcl defines the `unknown` command to use the following heuristic:

(a) See if the Tcl autoload facility can locate the command in a Tcl script file. If so, load it and execute it.

(b) If running interactively see if the command exists as an executable UNIX program. If so, `exec` the command.

(c) If the command was invoked at the top-level:

  (i) See if the command requests `csh`-like history substitution in one of the common forms `!!`, `!`*`number`*, or `^`*`old`*`^`*`new`*. If so, emulate `csh`'s history substitution.

  (ii) See if the command is a unique abbreviation for another command. If so, invoke the command.

(d) Otherwise, fail with an error indicated an unrecognized command.

This means that from the `dcecp>` prompt, a user can type `ls` to see a listing of the files in the current directory, or `rm foo` to delete the file `foo` (see about filename completion in the *Command-line Editing Commands* section).

8.3.1. Object-operation vs. operation-object

The choice of using an object-operation command order has been controversial. Many parties have stated that an operation-object ordering is more intuitive and is more common in other user interfaces. The major advantage to using an object-operation model is extensibility. To add a new object type to `dcecp`, that object type merely needs to be added with all supported operations. If an operation-object order was used, then each operation that was supported by the object would need to be taught about the new object. This would be a big problem as operations are expected to be common among objects, e.g., `create`, `delete`, and `show`.

In deference to the operation-object faction, `dcecp` will include a script that can be loaded by any user, or configured to be loaded for an entire system. This script will define commands named for operations, e.g., `create`, `delete`, and `show`. The commands will expect their first argument to be the name of an operation, and will try to execute that operation on that object using the object-operation syntax. This will provide an operation-object syntax to the user *in addition to* the object-operation order. The commands will be based on the Tcl `info commands` command and the `operations` command supported by each object. Therefore, as new objects are added to the system, these commands will learn about them automatically. A sample (and incomplete) implementation of the `show` command is shown below:

```
proc show {obj args} {
  if {[llength [info commands $obj*]] < 1} {
    error "Object $obj does not exist"
  } elseif {[llength [info commands $obj*]] > 1} {
    error "$obj not unique"
  } else {
    if  {[llength [lsearch [$obj operations] show]] < 1} {
      error "show command not supported by $obj object"
    } else {
      $obj show $args
    }
  }
}
```

This example will be modified to do somewhat better error checking. Note that the user of these commands will not have to understand the above script. All they will need to do is have the following command in their `.dcecprc` file or in the system's `init.dcecp` file:

```
source verb-object.dcecp
```

Also note that the object-operation order is the supported syntax. The documentation will show sample commands using that order. There will be a description of how to use the operation-object order. Since the difference is just transposing the first two words, this should not be a problem for an administrator of DCE choosing to use an operation-object syntax.

8.3.2. Attribute lists and options

Many of the commands need to specify attributes to operate upon. For example, the `modify` command allows attributes to be changed and the `create` command often allows attributes to be created along with the object. In all cases, a mechanism exists so that an attribute list is used to specify the attributes and their values. This makes passing information from one command to another very easy. For example, an ACL copy operation could be written as follows:

```
# copy acl name1 to acl name2
# no error checking
proc acl_copy {name1 name2} {
  acl replace $name2 -acl [acl show $name1]
}
```

While attribute lists are useful for writing scripts, they are often not user-friendly. For those objects that have a fixed list of attributes (e.g., `principal` and `dts`, but not `object`), wherever an attribute list is allowed, options for each attribute that are named the same as the attribute are allowed followed by their values. For example, the following are equivalent:

```
principal create melman -attribute {{quota 5} {uid 123}}
principal create melman -quota 5 -uid 123
```

8.3.3. Lists of lists

Since Tcl lists are confusing enough, it's worth noting here a common usage when lists contain lists. See the *Data Structures* section above for a brief description of Tcl lists, or see [TCL BOOK] for more detail. The following example is the command to remove some ACL's from a the object `/.:/foo`:

```
acl modify /.:/foo -remove {user melman}
```

The argument to the `-remove` option is an ACL Entry. The ACL Entry happens to be a list where the first element describes the ACL Type, in this case `user`, and the second is the key for which user, in this case `melman`. However, the `-remove` option may take a list of ACL Entries, so the following is valid as well:

```
acl modify /.:/foo -remove {{user melman} {user salamone}}
```

All of this seems pretty clear. The point to remember is that lists of one value that do not contain spaces, do not need braces. The string syntax of an ACL Entry allows the type and key to be separated by a `:`, so the following are valid:

```
acl modify /.:/foo -remove user:melman
acl modify /.:/foo -remove {user:melman user:salamone}
```

A point of confusion is if there is only one ACL Entry given, i.e., the `-remove` option's value has only one element (and that element does not contain spaces), then braces are not needed to delimit the list. The following are all valid, but are all examples with unnecessary braces:

```
acl modify /.:/foo -remove {{user melman}}
acl modify /.:/foo -remove {user:melman}
acl modify /.:/foo -remove {{user:melman} {user:salamone}}
```

The following is not legal since it is a list of a list of a list, and the value of the `-remove` option is a list of lists:

```
acl modify /.:/foo -remove {{{user melman}}}
```

Also note that lists are just strings with spaces and that braces are just one way to quote the spaces from being interpreted as argument separators, double quotes and backslashes would work as well. The following are all valid:

```
acl modify /.:/foo -remove {"user melman" "user salamone"}
acl modify /.:/foo -remove user:melman\ user:salamone
```

8.4. Abbreviations

The `dcecp` makes use of two mechanisms to allow all object names, operation names, and options to be abbreviated to the shortest unique string. The first is a standard Tcl mechanism builtin to the Tcl `unknown` command. See the *Command Syntax* section above.

Note that this mechanism only works if the command is entered interactively. If the command is found in a script, abbreviation checking is not performed by the standard implementation of `unknown`. This is to discourage the practice of using abbreviations of commands in scripts.

The other mechanism used for abbreviations is builtin to the individual `dcecp` commands themselves. They all share the same parser code which is used for both the operation names and the option names. This allows the operation name to be abbreviated to the shortest unique operation supported for that object, and the options to be abbreviated to the shortest unique string representing a option supported by that object and operation. This fact has had a bearing on the names of operations and options. For example, in most cases `-member` can be abbreviated `-m` since few other options begin with ‘‘m’’. Note that this form of abbreviation is always available, whether invoked interactively or via a script.

8.5. Convenience Variables

All `dcecp` commands will set several variables on execution. The variables will contain the name of the object operated on, the return value of the last command, the DCE name of the user, etc. Users can substitute the value of these variables into the next command to save typing. These are regular Tcl variables, so any mechanism to perform the variable substitution will be supported. The most common method is to prepend a `$` before the name of the variable, though the `set` command can be used as well.

All convenience variables are named with two characters: a leading underscore and a single letter. Currently all are lowercase, though it should be noted that Tcl is case sensitive. Also, one of the variables is an array so a subscript must be given with it. Some of these variables are read-only to the user (implemented with Tcl write traces), this is explicitly mentioned below.

The following variables are defined in `dcecp`:

(a) `_b` — holds the name of the server bound to for the last command. This is actually a Tcl array where the indexes are used to identify the service. Currently there is only one defined index: `sec`.

The value specifies the name of a server in whatever manner the service finds useful. This could be the name of an RPC server entry in the namespace, or a string binding, or the name of a cell. This variable may *not* be set by the user.

(b) `_c` — holds the name of the cell that the principal in the current login context is registered in. See the `_u` convenience variable below. This variable is read-only, setting it via `set` will generate an error.

(c) `_h` — holds the DCE name of the current host. This variable is read-only, setting it via `set` will generate an error.

(d) `_n` — holds a list of the names entered to the last object command.[8] These names are the names that the command operated on, typically entered as the third argument. The names stored in the variable are listed exactly as they were entered in the command, they are *not* converted to canonical form. Examples follow:

```
dcecp> dir list /.: -simplename
hosts subsys absolut_ch cell-profile fs lan-profile sec \
sec-v1
dcecp> echo $_n
```

8. Note that this means the last object command, not any Tcl command or miscellaneous `dcecp` commands like `login`.

```
/.:
dcecp> dir create {/.:/x /.:/y}
dcecp> echo $_n
/.:/x /.:/y
```

(e) `_o` — holds the object used in the last operation. For example, if the last command was `dir show /.:`, then `$_o` is `directory`. This variable is read-only, setting it via `set` will generate an error.

(f) `_p` — holds the parent of `_n`. If `_n` is a list, then this is a list of the same length, where each element is the parent of the corresponding element in `_n`. This is accomplished syntactically by removing everything before the last `/`. If there is no `/` in `_n`, `_p` is an empty string. This variable is read-only, setting it via `set` will generate an error.

(g) `_r` — holds the return value of the last executed command. This is the return value of *any* command given to `dcecp`, not just those returned by object commands. This is different than the behavior of `_n`. This variable is read-only, setting it via `set` will generate an error.

(h) `_s` — holds the name of a server to bind to for the next command. This is actually a Tcl array where the indexes are used to identify the service. The currently defined indexes are: `sec`, `cds`, `dts` and `aud`.

The value specifies the name of a server in whatever manner the service finds useful. This could be the name of an RPC server entry in the namespace, or a string binding, or the name of a cell. This variable may be set by the user, it is not set by `dcecp`.

The values of this variable (array) are treated differently by each service. For example, the security service uses this variable as a default for the next registry operation. If bound to a read-only replica and an update is requested, `dcecp` will try to bind to the master registry to perform the change. The CDS service uses this variable in the same way as the `cdscp` concept of a `preferred clearinghouse`. If set, CDS will only attempt to communicate with the specified server (the word ''preferred'' was a poor choice). The auditing service uses this variable in a similar manner to the CDS server. To contact an audit daemon on another host, set this variable to identify that server. The DTS service behaves the same way.

If any objects set or read this value, their behavior is explicitly mentioned in the description of the object in following sections.

(i) `_u` — holds the current simple principal name. This variable can be set by the user to change the current login context. See the `login` command. The cellname of `_u` is stored in `_c`, so the fully qualified principal name is `$_c/$_u`. This variable is read-only, setting it via `set` will generate an error.

8.6. Error handling

All commands in `dcecp` return either a list of some information (e.g., the `show` and `list` operations), or an empty string on success.

The `dcecp` will make use of Tcl's native error handling facilities. At the C code level, all Tcl commands return a code and a string. The code is usually either `TCL_OK` or `TCL_ERROR`. The returned string is either the normal result of the command or the error message. Tcl stores the stack-trace of nested errors, and this is accessible to the user. Tcl also provides facilities for scripts to catch errors and invoke error handlers (errors are one of four types of exceptions that Tcl supports).

Also there are two global Tcl variables visible in `dcecp`, `errorInfo` and `errorCode`. The former contains the stack-trace of the error messages, and the later is meant to be machine readable information. The `errorCode` variable is a relatively new addition to Tcl so its use is limited. Currently the commands that do I/O and such set `errorCode` to be a list where the first element is `POSIX` and the next two elements are the Posix name for the error, such as `ENOENT` and a human-readable message describing the error.

On error, `dcecp` commands will return the message string of the error, the Tcl Error exception (which can be caught with the Tcl `catch` command), and will set `errorCode` as appropriate to a list where the first element is `DCE` and the second is the numeric value of the status code. The main loop of `dcecp` will print the text of any error to standard error. If the global variable `dcecp_verbose_errors` is set to `1`, `dcecp` will instead output the entire contents of `errorInfo` with the string `Error:` prepended. A `dcecp` script which traps an error is shown below:

```
# This routine returns 1 if the argument is the name
# of a CDS directory, 0 if it is not.  It uses the show
# operation to determine if it is a directory.  If the
# command works then it is a valid directory, if not,
# an error is generated and caught by the Tcl catch
# command and the routine returns 0.

proc isdirectory {name} {
  if {[catch {directory show $name}]==0} {return 1} {return 0}
}
```

In cases where an argument list is given to a command indicating that an operation is to be performed on more than one object, the operations are usually performed iteratively. If there is an error, the command will abort at the time of error with the Tcl `TCL_ERROR` exception. Some operations will have finished and others will not have. The operations are always performed in the order entered and the error message should make it clear on which object the command failed.

8.6.1. Warning Messages

Conspicuously absent from `dcecp` are any warning or ''Are you sure?'' type messages. This is intentional. The rationale is based on the scriptability requirements of `dcecp`. All the commands either succeed or fail with an error message. Extra output to `stderr` or `stdout` does not occur. Only what is returned to the Tcl interpretor is displayed to `stdout`. A user writing a script can depend on everything that is seen on `stdout` being returned by the commands.

The `dcecp` commands never prompt for input (with one exception described below). All the commands complete individually. This also improves scriptability. The one exception to this rule are some commands that require a password to be entered (e.g., `login`). In this case a password is prompted for and echoing is turned off. All of these commands accept a command-line option to enter the password so that scripts can be easily written.

Typically user-friendly ''Are you sure?'' messages are included in the above principle. They are difficult to write scripts around. However, these same scripting capabilities allow users to write ''wrappers'' around any command that would ask an ''Are you sure?'' question and proceed only after a positive response. These are all left as an exercise to the user or vendor, however, users should always be able to access the commands described in this specification without fear of invoking ''Are you sure?'' questions.[9]

8.7. Help System

There will be several different kinds of help available to the user of `dcecp`. All will return help strings obtained from message catalogs as appropriate.

Each `dcecp` object will support the `operations` command. This returns a list of the operations supported by the object. It is similar to the Tcl builtin command `info commands`. An example is:

```
dcecp> principal operations
create delete modify show catalog operations help
```

This provides only very limited help, but it is important that the output is machine readable (i.e., a well formed Tcl list). It is analogous to a simple usage message found on many systems. Users unsure of an operation name or if an operation is supported by an object can use this command to find the answer.

Each `dcecp` object will also support a `help` command. This command has three different modes. If called with no arguments or options, it returns a one line per operation help message. Operations are listed in alphabetical

9. Perhaps a new environment variable `dcecp_confirmations` could be added to specify this behavior without having to create entirely new commands.

order, with the `operations` and `help` commands listed last since they are supported by all `dcecp` objects. An example is:

```
dcecp> principal help
catalog     Return the names of all principals in the registry
create      Creates a new principal
delete      Deletes the named principal
modify      Changes attributes of a principal
show        Return all the attributes of a principal
help        Provides help on principals
operations  Returns names of operations supported by principal
```

The `help` command takes one argument which is interpreted as the name of one operation[10], (not a list of operations). If invoked in this manner, a one line per option help message is returned with information on each option supported by the command. Attribute options are listed in alphabetical order, other options follow in alphabetical order as well. If no options are supported then an empty string is returned. An example is:

```
dcecp> principal help create
-alias      If present principal is an alias (not a primary)
-fullname   PCS limited description of principal
-quota      Maximum number of principals this principal can
            create
-uid        The User ID of the principal (can be generated)
-uuid       Automatically generated internal UUID of the
            principal
-change     Give a list of attributes and new values for
            them
```

Finally, if the `help` command is called with the `-verbose` option, then a one paragraph description of the object is returned. This command will return text explaining what the object represents and how to use it. The level of information returned will be on the order of the first paragraph of the DESCRIPTION section of a manual page. This description should include a definition of what the object command represents, the possible operations and attributes (or at least the commonly used ones), perhaps permission information, side effects, etc.

```
dcecp> principal help -verbose
The principal object represents Registry principals.  All of
the principal operations take one argument which is a list
of names of principals to act on.  It must be a principal
name, not the name of the database object that contains the
registry information about the principal (i.e., it should
not begin with /.:/sec/principal/).  After this command
executes, the _b(sec) convenience variable is set to the name
of the server that was bound to for the command.  The value
of the variable before the command is treated as a hint.
In the registry, each principal has a primary name and a
value for each of the following attributes: alias, fullname,
quota, uid, uuid.
```

Also, due to the fact that the Tcl `unknown` command does a `PATH` search for commands that it does not know about (see the *Command Syntax* section), a user can get detailed help on an object by invoking the `man` command

10. The word order is a bit strange here, but the rationale is as follows. This order creates a real `help` operation which behaves similarly to other operations. It takes the name of an operation to act on, much as most commands take the name of some object instance to act on. Alternative orderings such as `principal create help` have the problem that it is then impossible to create a principal of the name `help`. Also the specified order allows a `help` verb operation (see the *Object-operation vs. operation-object* section) to be created which supports the `help principal create` syntax.

on a [POSIX.2] system specifying an object name as the argument to `man`. For example, the user can get detailed help on the `principal` command by entering:

```
dcecp> man principal
```

8.8. Internationalization

Areas that need to be addressed by internationalization include:

(a) `dcecp` scripts will be limited to run in the locale in which they were written. All scripts shipped by OSF will be in the C Locale.

(b) Portions of Tcl will need to be internationalized:

(i) Basic parsing routines, since the only datatype in Tcl is a string.

(ii) Most output strings (errors but not prompts) will need to be isolated into message catalogs.

(iii) The `scan`, `string`, `lsort`, `lsearch`, and `lreplace` commands.

(iv) The regular expression handling.

(v) The `glob` command.

8.9. Common operations

This section gives a description of operations that are common to more than one object. Some operations presented here are implemented in all objects, some in only a few, and some only for specific types of objects such as containers. Each object is described fully in a following section. Each of those sections lists each operation that the objects supports. Unless otherwise mentioned in those descriptions, those given here apply for the operations.

This section also defines the arguments for the operations. Often the phrase ''a list of names'' is used. Please note that one name is the degenerate case of a list of names, and that in Tcl, braces are only required on lists of more than one item. For example the `delete` operation is described below as taking a list of names as arguments. Both of the following are valid:

```
object delete {foo bar}
object delete foo
```

The descriptions in the sections on individual objects may override some of the information presented here. Usually this is only in the form of an operation accepting more options, but other changes are possible.

(a) `add`

Adds an object to a container. It is implemented for all objects that represent containers. Returns an empty string on success. The argument is a list of names of containers. The required `-member` option is used to specify the name of the member to be added to the containers. Its value is a list of members to be added. If lists are specified for both the `-member` option and as the argument, then each member name is added to each container.

For example it is used to add a member to an RPC group, and is used to add an element to an RPC profile.

(b) `catalog`

Returns the names of all instances of an object. It usually takes no arguments but sometimes takes a cell name. For example, the `principal catalog` command returns a list of all principals in the registry. Only implemented by those objects for which this is possible. By default, fullnames are returned. Some objects will support a `-simplename` option which will return names in a shorter form (either relative or not fully qualified). The order of the returned list is dependent on the service.

(c) `create`

Creates a new instance of an object. Takes one argument which a list of names of instances to be created. Returns an empty string on success. Returns an error if the object already exists. For some objects this command takes a `-attribute` option or a set of attribute options to create attributes on the new object.

(d) `delete`

Destroys an instance of the object. Takes one argument which a list of names of instances to be deleted. Returns an empty string on success. If the object does not exist, an error is returned.

(e) `help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `list`

Returns a list of the names of all the members of a container. This operation only returns names of members, never any other (e.g., attribute) information about the members. It is to be implemented for all objects that represent containers. The argument is a list of names of containers to return the children of. The order of the returned list is dependent on the service. If more than one container name is given as an argument, all the member names are returned in one list. The user may always use the output of this command as a list in a `foreach` statement. For example:

```
foreach i [dir list /.: -objects] {object show $i}
```

The names returned are fully qualified names by default. In some objects, the member's names are the same as the parent's with another simple name (RDN in X.500 terminology, name component in other terminology, everything following the last `/` in others) appended to the end. In others, such as RPC Profiles, the members must be fully qualified global names. Since some names *must* be fully qualified, the `list` operation by default returns all names as fully qualified.

Some objects implement the `-simplename` option which indicates the command should return simple names, i.e., names relative to the parent. It will be implemented for all objects that can meaningfully return simple names; e.g., `rpcprofile`'s will not implement this, but `directory` will.

(g) `modify`

This operation is used to modify attributes, policies, counters, or any other information in an object. This fact means that all attributes, policies, counters, etc. in any one object must have unique names. It will not be available to all objects. Returns an empty string on success. The argument is a list of names of objects to modify. All objects are modified in the same way.

The specific modification is described by the use of one or more of the following options. If more than one is used, then the whole modify operation is treated atomically in that either it all will work, or none of it will. The order of the options does not matter. Each option may only be used once per command invocation. If more then one attribute is to be added, then the value of that option should be an attribute list.

(i) `-add`

Used to add an attribute to an object or merely to add values to an existing attribute. The value of this option is an attribute list. If only an attribute type is specified after the option, then it is added to the object with no values (if the object permits this). If an attribute type and value(s) are given, then they are added to the object, with the attribute being created if necessary. If the attribute or value is already present an error is returned. This error implies that if the same attribute is listed twice in the same command, an error is generated.

(ii) `-remove`

Used to remove an entire attribute or merely some values from an object. The value of this option is an attribute list. If only the attribute type is specified after the option, then the entire attribute is removed. If an attribute type and value is specified then only that value is removed. If it is the only value present in the attribute then the entire attribute is removed. If the attribute or value is not present an error is returned. This error implies that if the same attribute is listed twice in the same command, an error is generated.

(iii) `-change`

Used to change one attribute value to another. The value of this option is an attribute list. Each attribute in the list has its existing value replaced by the new value given in the attribute list. For multi-valued attributes, *all* existing values are removed, and replaced by all the values listed for the attribute in the specified attribute list. For some objects this function is equivalent to doing a `-remove` followed by an `-add` for the same attribute. If the attribute or value is not present an error is returned. If the same attribute is listed twice, both changes are made, with the second change being the final result.

Some objects implement attribute options in addition to attribute lists as described above in the section *Attribute Lists and Options*. When this is done either attribute lists or attribute options may be used on a single command line, but not both.

(h) `operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last. If the user wants them sorted they should use:

```
lsort [object operations]
```

All operations valid for the object are returned, including the `operations` operation.

(i) `remove`

Removes an object from a container. For example, it is used to remove a member from an RPC group or an element from an RPC profile. It is implemented for all objects that represent containers. The argument is a list of names of containers.

Requires one option, `-member`, which is used to specify the name of the member to be removed from the container. The value is a list of names of members of the containers. If the value of this option and the argument to the command are both lists, then each listed member is removed from each specified container. If the members do not exist an error is returned.

(j) `rename`

This operation changes the name of a specified object. The argument is a single name of an object to be renamed, i.e., it cannot be a list. Takes a required `-to` option with a value of the new name. The value may not be a list. Returns an empty string on success.

(k) `show`

Returns information about an object instance. Objects can have various types of information such as attributes, counters, policies, etc. The `show` command is used to return any of this information. Options are passed to the command to specify what information is to be returned. All of the options used for this purpose are in the plural form. Each object defines a default option, defining what information is to be displayed when no options are passed. In almost all cases this will be `-attributes`, and unless specified otherwise, this should be assumed.

Unlike the `list` command which returns information about the members of a container, the `show` command only looks at the named object instance. If it is a container, it does *not* return information about the members, only the container itself. However, sometimes a container contains more than just the name of a member. For example, an RPC profile is a container that has no attributes, counters, or anything else. However, for each member there is additional information such as an annotation and a priority. This information is not in the member objects, only in the container stored with the reference to the member. This member information (or child pointer information in CDS parlance) can be displayed with the `show` command.

This command takes one argument which is a list of names of instances to be shown. If more than one name is included, the output is organized as if the command was invoked separately for each instance. The attributes (or other information) for each instance is displayed in a list, all captured as one element in a larger list. The common options to the `show` command are:

(i) `-all` — Return a combination of all showable information.

(ii) `-attributes` — Return information about attributes.

(iii) `-counters` — Return information about counters.

(iv) `-members` *name* — Return information about the member identified by *name* that is stored in the container. In some cases *name* need not be specified in which case all information that is stored in the container about all members is returned.

(v) `-policies` — Return information about policies.

The output of the command is in the form of an attribute list. If different types of information are specified to be returned by the use of multiple options or the `-all` option, then the attribute lists are concatenated together. All attributes, counters, and policies are named from the same namespace and must be unique. For example, the `organization show` *orgname* `-all` command would return a list of the form:n

```
{fullname xxxx}
{orgid xxxx}
{uuid   xxxx}
{acctlife xxxx}
{pwdalpha xxxx}
{pwdexpdate xxxx}
{pwdlife xxxx}
{pwdminlen xxxx}
{pwdspaces xxxx}
```

Attributes (or other data) returned by the `show` operation are returned in an order defined by the object. For example, CDS attributes are returned in lexical order of the string representations of OID's of the attribute types, while registry commands return fixed data in fixed order and ERA information in an unpredictable order.

(l) `synchronize`

Tells the instance to synchronize with any replicas of itself. In CDS terminology this will perform a skulk on a directory, in DTS it will cause a server to synchronize. It is to be implemented for all objects that support replication. Returns an empty string on success. The argument is a list of instance names to synchronize. If more than one instance name is given then each instance synchronizes, there is no relationship such as synchronize with each other, as this doesn't make sense for many objects.

8.10. Common options

The following are options that are implemented for some commands. When used, they behave in a common manner, although there may be slight differences in the implementations.

(a) `-annotation`

Used to specify an annotation string.

(b) `-attribute`

Used to specify an attribute consisting of a type and one or more values. Attributes are represented as a list where the first element is the type and the following elements are the values. Where applicable (e.g., in CDS) attribute types are specified, with either OID's or their string representation.

(c) `-binding`

Used to specify a string binding.

(d) `-default`

Used to specify that an operation is to be invoked on the default profile member (element). When this option is specified other options used to identify a particular profile member are illegal.

(e) `−interface`

Used to specify an interface identifier.

(f) `−member`

Used to identify an item in a container.

(g) `−object`

Used to specify an object UUID.

(h) `−version`

Used to specify how the version specification of the interface identifier given in the `−interface` option is used. Requires a value which is one of the following:

(i) `all`

The interface version is ignored, all values are acceptable. This is the default value.

(ii) `exact`

Both the major and minor versions must match the specified versions.

(iii) `compatible`

The major version must match the specified version, and the minor version must be greater than or equal to the specified version.

(iv) `majoronly`

The major version must match the specified version; the minor version is ignored.

(v) `upto`

The major and minor versions must be less than or equal to those specified.

8.11. Objects

The following are the object commands supported by `dcecp`. The supported operations are listed for each command. Users can add more objects by using the Tcl `proc` command to define a new command. The tasks described in the next section are examples of doing this.

8.11.1. rpcentry

The `rpcentry` object represents an RPC server entry. The argument is a list of names of server entries. Each server entry contains a set of object UUID's, and a set of interface identifier/string binding pairs. Supports the following operations:

(a) `rpcentry create`

Creates an empty entry. Since an empty entry is the same as an empty RPC group or RPC profile, calling `rpcentry create` is the same as calling `rpcgroup create` or `rpcprofile create`. The argument is a list of names of RPC entries to be created. Returns an empty string on success. If the RPC entry already exists, an error is returned.

(b) `rpcentry delete`

Deletes the specified entry. The argument is a list of names of server entries to be deleted. Returns an empty string on success. If the entry does not exist an error is returned.

(c) `rpcentry export`

Exports binding information to the specified entry. The argument is a list of names of server entries to be exported to. If an entry does not exist, it is created. Uses the `−interface`, `−binding`, and `−object` options to specify what to export. Returns an empty string on success.

This operation will also be able to export codeset and other attributes as specified in the RPC Internationalization specifications.

(d) `rpcentry help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `rpcentry import`

Returns a string binding from the specified RPC entry. The argument is a list of names of RPC entries to import from. Uses the `-interface`, `-version`, and `-object` options to specify matching bindings. Each of these takes only one value not a list of values. Also accepts the `-max` option to specify a number of string bindings to return. If the value is greater than one, then a list of as many matching bindings less than or equal to the value is returned. Reports an error if there was no match in any of the entries. The `-noupdate` option is used to avoid updating a local cache before performing the operation. The order of bindings returned is arbitrary.

This operation will also be able to distinguish acceptable entries based on codeset and other attributes as specified in the RPC Internationalization specifications.

(f) `rpcentry operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `rpcentry show`

Returns a list containing the binding information in the RPC server entry. The list is composed of two lists. The first list is also a list, where the first two elements are the interface identifier (the uuid and then the version), and the rest of the elements are string bindings in Tcl syntax. The second list is a list of Object UUID's exported by the server. The order of the data returned is arbitrary. The argument is a list of names of server entries to return information about. For example:

```
dcecp> rpcentry show /.:/subsys/dce/sec/master
{0d7c1e50-113a-11ca-b71f-08001e01dc6c 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{d46113d0-a848-11cb-b863-08001e046aa5 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{5b8c2fa8-b60b-11c9-be0f-08001e018fa0 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{b1e338f8-9533-11c9-a34a-08001e019c1e 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{47b333318000.0d.00.01.dc.6c.00.00.00 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{8f73de50-768c-11ca-bffc-08001e039431 1.0
  {ncacn_ip_tcp 130.105.54.93}
  {ncadg_ip_udp 130.105.54.93}}
{6bb55e14-459e-11cd-ab9c-080009251352}
```

This operation recognizes the `-interface` option (its value is a list of interface id's) so that only string bindings matching that interface identifier are returned. The match is exact. The `-object` option is used to specify a list of object UUID's to match. The `-noupdate` option is used to avoid updating a local cache before performing the operation.

This operation will also return codeset and other attributes as specified in the RPC Internationalization specifications.

NOTE: Details of syntax are TBD.

(h) `rpcentry unexport`

Removes binding information from an entry. The argument is a list of names of server entries to have information unexported from. Uses the `-interface`, `-version`, and `-object` options to specify what to unexport. Each can have only one value, not a list of values. Returns an empty string on success. An error is generated if the information is not found.

8.11.2. rpcgroup

The `rpcgroup` object represents an RPC group. Each RPC group is named in the DCE namespace; therefore each operation described below takes one argument which is the name of the group to act upon. An RPC group is a container of names of either RPC server entries or other RPC groups; there is no data in the group other than the name of the members. Supports the following operations:

(a) `rpcgroup add`

Adds a member to the specified group. The argument is a list of names of RPC groups to have members added to. The value of the required `-member` option is a list of names which are references to an RPC entry, RPC group, or RPC profile (i.e., they do not have to actually exist). All members are added to each group mentioned in the argument list. Returns an empty string on success. If the new members already exist, duplicates are added to the RPC group.[11]

(b) `rpcgroup create`

Creates an empty group. Since an empty group is the same as an empty RPC entry or RPC profile, calling `rpcgroup create` is the same as calling `rpcentry create` or `rpcprofile create`. The argument is a list of names of RPC groups to be created. Returns an empty string on success. If the RPC group already exists, an error is returned.

(c) `rpcgroup delete`

Deletes the specified group. The argument is a list of names of RPC groups to be deleted. Returns an empty string on success. If the RPC group does not exist an error is generated.

(d) `rpcgroup help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `rpcgroup import`

Returns a string binding from the specified RPC group. The argument is a list of names of RPC groups to import from. Uses the `-interface`, `-version`, and `-object` options to specify matching bindings. Each of these takes only one value not a list of values. Also accepts the `-max` option to specify a number of string bindings to return. If the value is greater than one, then a list of as many matching bindings less than or equal to the value is returned. Reports an error if there was no match in any of the groups. The `-noupdate` option is used to avoid updating a local cache before performing the operation. The order of bindings returned is arbitrary.

This operation will also be able to distinguish acceptable entries based on codeset and other attributes as specified in the RPC Internationalization specifications.

---

11. This is the behavior of the NSI.

(f) `rpcgroup list`

Returns a list of the names of all members of the specified group. The returned names are fully qualified names and are returned in an arbitrary order. The argument is a list of names of RPC groups to have their members names returned. The `-noupdate` option is used to avoid updating a local cache before performing the operation. If called so that more than one RPC group is listed then the members are concatenated on output into one list with a blank line between RPC groups.

(g) `rpcgroup operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `rpcgroup remove`

Removes one or more members from the specified group. The argument is a list of names of RPC groups to have members removed from. The value of the required `-member` option is a list of names of RPC entries, RPC groups, or RPC profiles; they are only references stored in the RPC group and does not have to actually exist outside of the group. All members specified are removed from all RPC groups specified in the argument. Returns an empty string on success. If a specified member does not exist in an RPC group an error is returned.

8.11.3. rpcprofile

The `rpcprofile` object represents an RPC profile. Each operation described below takes one argument which names the RPC profiles to be acted upon. An RPC Profile consists of members (also known as elements in other DCE documentation) which can be either RPC server entries, RPC groups, or other RPC profiles; therefore each member of a profile has a name in the DCE namespace. Each profile can have one member which is the default member of the profile (a.k.a. the default profile element).

A profile contains no attributes, but it does contain information about each member, which is not contained in the member itself. This information is an interface identifier, a priority, and an optional annotation. They are identified by the following keywords: `interface`, `priority`, and `annotation`. The keyword `member` is used to identify the member name.

Supports the following operations:

(a) `rpcprofile add`

Adds a member to the specified profile. The argument is a list of names of RPC profiles to have members added to. The value of the required `-member` option is a list of names which are references to an RPC entry, RPC group, or RPC profile (i.e., they do not have to actually exist). Accepts the `-interface`, `-priority`, and `-annotation` options with one value (not a list) each. All members are added to each profile mentioned in the argument list. Also accepts a `-default` option to indicate that the add operation is performed on the default profile member (other options are illegal if the `-default` option is used). Returns an empty string on success. If the new members already exist, an error is returned. If the profile does not yet exist, it is created.

(b) `rpcprofile create`

Creates an empty profile. Since an empty profile is the same as an empty RPC entry or RPC group, calling `rpcprofile create` is the same as calling `rpcentry create` or `rpcgroup create`. The argument is a list of names of RPC profiles to be created. Returns an empty string on success. If the RPC profile already exists, an error is returned.

(c) `rpcprofile delete`

Deletes the specified profile. The argument is a list of names of RPC profiles to be deleted. Returns an empty string on success. If the RPC profile does not exist an error is generated.

(d) `rpcprofile help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `rpcprofile import`

Returns a string binding from the specified RPC profile. The argument is a list of names of RPC profiles to import from. Uses the `-interface`, `-version`, and `-object` options to specify matching bindings. Each of these takes only one value not a list of values. Also accepts the `-max` option to specify a number of string bindings to return. If the value is greater than one, then a list of as many matching bindings less than or equal to the value is returned. Reports an error if there was no match in any of the profiles. The `-noupdate` option is used to avoid updating a local cache before performing the operation. The order of bindings returned is arbitrary.

This operation will also be able to distinguish acceptable entries based on codeset and other attributes as specified in the RPC Internationalization specifications.

(f) `rpcprofile list`

Returns a list of the names of all members of the specified profile. The returned names are fully qualified names and are returned in an arbitrary order. The `-noupdate` option is used to avoid updating a local cache before performing the operation. The argument is a list of names of RPC profiles to have their members names returned. If called so that more than one RPC profile is listed then the members are concatenated on output into one list with a blank line between RPC profiles.

(g) `rpcprofile operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `rpcprofile remove`

Removes one or more members from the specified profile. The argument is a list of names of RPC profiles to have members removed from. The members to be removed are those that match the values given in the following options: `-member`, `-interface`, `-priority` and `-annotation`. The values of each of these options is a single value, not a list, any members that match any of these values are removed from all RPC profiles specified in the argument. The options are exclusive and may not be combined, except `-interface` and `-member` which must be used together. Also accepts a `-default` option, in which case, the above options are illegal and the default profile member is removed. Returns an empty string on success. If a specified member does not exist in an RPC profile an error is returned.

(i) `rpcprofile show`

Returns a list of all members of one or more profiles. The argument is a list of names of RPC profiles to have members of returned. An attribute list is returned for each member with all of the entered information. It is always in the order: `interface`, `member`, `priority`, `annotation`. If any of the items is not given, they are not included in the output, i.e., no place holder is included.

Only those members that match the values specified by the following optional options are returned: `-interface`, `-version`, `-annotation`, `-member`. Each option may have only one value (i.e., the value may not be a list). Also accepts a `-default` option, in which case, the above options are illegal and the default profile member is returned. The `-noupdate` option is used to avoid updating a local cache before performing the operation.

8.11.4. endpoint

The `endpoint` object represents RPC endpoint mappings. Endpoint mappings contain an interface identifier and one or more string bindings; optionally they contain object UUID's and an annotation. Endpoint mappings are stored in the endpoint mapper. In DCE 1.0 this was implemented in `rpcd`, in DCE 1.1 this is implemented in `dced`. The `server` object has some operations (e.g., `disable` and `enable`) that affect endpoints stored in `dced`, but those operations will not work with `rpcd`'s. The `endpoint` object affects all endpoint maps, whether implemented in `rpcd` or `dced`.

Since endpoints have no names, the argument to these commands is not the name of an endpoint. Earlier versions of `rpccp` and `rpcd` allowed remote access to endpoints, but this was a security problem. The `endpoint` object does not allow remote access to endpoint maps. As such, the commands accept no arguments. The `server` object allows some remote operations on `dced` endpoint maps which are free of the security problem. Supports the following operations:

(a) `endpoint create`

Creates new endpoints in the endpoint map database. This command takes no arguments. Requires the `-interface` and `-binding` options. Also accepts the `-object` and `-annotation` options. The value of the `-binding` and `-object` options can be a list, but the others must be a single value. This command creates a cross product from the `-interface`, `-binding` and `-object` options and adds each element in the cross product as a separate registration in the local endpoint map. If you supply no object UUID's, the corresponding elements in the cross product contain a nil object UUID. If the mapping already exists it is replaced unless the `-noreplace` option is used. Returns an empty string on success.

For example, suppose that you have an interface `if1`, three bindings, `b1`, `b2`, and `b3`, and four object UUID's, `o1`, `o2`, `o3` and `o4`. The resulting 12 elements in the cross product are as follows:

```
{if1,b1,o1}  {if1,b1,o2}  {if1,b1,o3}  {if1,b1,o4}
{if1,b2,o1}  {if1,b2,o2}  {if1,b2,o3}  {if1,b2,o4}
{if1,b3,o1}  {if1,b3,o2}  {if1,b3,o3}  {if1,b3,o4}
```

An annotation string is part of each of these 12 elements, but is not shown for clarity.

(b) `endpoint delete`

Deletes the specified endpoints from the endpoint map database. This command takes as an optional argument a single string binding to an endpoint mapper to contact to service the command. Requires the `-interface` and `-binding` options; and accepts the `-object` option. The value of the `-binding` and `-object` options can be a list, but the others must be a single value. This command creates a cross product from the `-interface`, `-binding` and `-object` options and removes each element in the cross product from the local endpoint map. See `endpoint create` above for more details. If the mappings don't exist an error is generated. Returns an empty string on success.

(c) `endpoint help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `endpoint operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(e) `endpoint show`

Returns a list of information about endpoints. With no options returns all the endpoint mappings. The `-interface`, `-version`, and `-object` options can be used so that only those endpoint mappings matching the values of the options are returned. The `-object` option accepts a list as a value; the others do not. This command takes as an optional argument a single string binding to an endpoint mapper to contact to service the command.

May also be called with the `-uuid` option which returns the UUID of the endpoint map database. Each database gets its own UUID on creation (e.g., whenever `dced` is started). This UUID is exported as an object UUID to the namespace. Knowing what the current UUID is, allows an administrator to delete any other (stale) UUID's which may be in the RPC entry. If the `-uuid` option is used, all other options are illegal.

8.11.5. object

The `object` object represents CDS objects. An object in the CDS directory service, has a name in the DCE Cell namespace, and a collection of attributes associated with it. Attributes consist of a type and one or more values. Every object is the child of a directory. See the `directory list` command for information on how to get a list of objects beneath a directory.

If the `_s(cds)` convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do *not* set the value of this variable after completion.

There is no concept of current working directory, or cd'ing through the CDS namespace. Certainly this is a possibility for future expansion.

The following are the CDS defined attributes that may be present in CDS Objects.

(a) `CDS_Class`

Specifies the class to which an object belongs.

(b) `CDS_ClassVersion`

Contains the version number of the object's class. This allows applications to build in compatibility with entries created by earlier versions.

(c) `CDS_CTS`

Specifies the creation timestamp (CTS) of this object entry. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users.

(d) `CDS_ObjectUUID`

Specifies a UUID for the object being referenced. The value of this attribute may only be specified on a `create` operation, thereafter it is read-only.

(e) `CDS_UTS`

Specifies the timestamp of the most recent update to an attribute of the object entry. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users.

Supports the following operations:

(a) `object create`

Creates a new object. The argument is a list of names of objects to be created. Takes an optional `-attribute` option to specify a list of attributes to be included in each created object. Added attributes are multi-valued unless a `-single` option is specified, in which case all attributes are single-valued. The `-single` option is only valid if the `-attribute` option is specified. Returns an empty string on success.

(b) `object delete`

Deletes an object. The argument is a list of names of objects to be deleted. Returns an empty string on success. An error is returned if any of the objects do not exist.

(c) `object help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `object modify`

Used to add or remove attributes and their values from the object. The argument is a list of names of objects to be modified. Accepts the `-add`, `-remove`, and `-change` options which each can have an attribute list as the value. See the *Common Operations* section for details. Attribute options are *not* supported for this command. Returns an empty string on success.

Since attributes in CDS can be multi-valued, some ambiguity can result in the syntax of some commands. To resolve this ambiguity, this command takes two additional options that describe the values to the `-add` and `-remove` options. Multi-valued attributes are considered to be the default case, when dealing with single-valued attributes the following options should be used:

(i) `-single`

If this option is present then the attributes specified in the `-add` option are to be single-valued. Normally all user defined attributes are defined to be multi-valued, even if only one value is specifed. This option is not legal without the `-add` option.

(ii) `-types`

If this option is present then the value of the `-remove` option is a list of attribute types, indicating that the entire attribute should be removed, not just a value. This option is not legal without the `-remove` option.

(e) `object operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `object show`

Returns an attribute list of the attributes in the specified objects. The argument is a list of names of objects to be shown. If more than one object is to be shown, the attributes of all the objects are concatenated into one list with a blank line between objects. The order of the returned attributes is the lexical order of the OID's of each attribute for each object. For example:

```
dcecp> object show /.:/obj
{RPC_ClassVersion
 {0200}
 {0300}}
{RPC_Group 1234}
{CDS_CTS 1994-07-01-22:06:54.990-05:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-07-01-22:07:37.248-05:00I0.000/00-00-c0-f7-de-56}
{CDS_Class 0200}
dcecp>
```

A `-schema` option can be used to return whether an attribute is single or multi-valued. This is specific to an object, i.e., the same attribute can be single-valued on one object and multi-valued on another object. This option may not be used with other options. For example:

```
dcecp> object show /.:/obj -schema
{RPC_ClassVersion multi}
{RPC_Group multi}
{CDS_CTS single}
{CDS_UTS single}
{CDS_Class single}
dcecp>
```

8.11.6. directory

The `directory` object represents CDS directories. CDS directories are containers for other objects, links, and other directories (as well as clearinghouses). If any of these items reside in a directory, they are said to be called children of that directory. Directories also contain attributes that may be viewed or modified.

This object also represents the CDS concept of replicas. Replicas are read-only copies of directories stored in other clearinghouses. Several of the supported operations take options to indicate that the command is to operate on a specific replica.

If the `_s(cds)` convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do *not* set the value of this variable after completion. If a `-clearinghouse` option is used (as described in some commands below), then it will override the value of `_s(cds)` but the command will *not* change the setting of `_s(cds)`.

The following are the CDS defined attributes that may be present in directories and replicas in CDS:

(a) `CDS_AllUpTo`

Indicates the date and time of the last successful skulk on the directory. All replicas of the directory are guaranteed to receive all updates whose timestamps are less than the value of this attribute. The value of this attribute is a DTS style timestamp. This attribute is written by the system and is read-only to users.

(b) `CDS_Convergence`

Specifies the degree of consistency among replicas. This attribute's value is defined as one of the following:

(i) `low`

CDS does not immediately propagate an update. The next skulk distributes all updates that occurred since the previous skulk. Skulks occur at least once every 24 hours.

(ii) `medium`

CDS attempts to immediately propagate an update to all replicas. If the attempt fails, the next scheduled skulk makes the replicas consistent. Skulks occur at least once every 12 hours.

(iii) `high`

CDS attempts to immediately propagate an update to all replicas. If the attempt fails (for example, if one of the replicas is unavailable), a skulk is scheduled for within one hour. Skulks usually occur at least once every 12 hours. Use this setting temporarily and briefly, because it uses extensive system resources.

By default, every directory inherits the convergence setting of its parent at creation time. The default setting on the root directory is `medium`.

(c) `CDS_CTS`

Specifies the creation timestamp (CTS) of the CDS directory. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users.

(d) `CDS_DirectoryVersion`

Specifies the current version of the directory (derived from the `CDS_DirectoryVersion` attribute of the clearinghouse in which the directory was created). Multiple directory versions are supported in a cell. This attribute is written by the system and is read-only to users.

(e) `CDS_Epoch`

A UUID that identifies a particular incarnation of the directory. This attribute is written by the system and is read-only to users.

(f) `CDS_GDAPointers`

A set-valued attribute which is only present in the root directory of a cell. This attribute contains location information about registered Global Directory Agents for that cell, similar to the CDS_Replicas attribute. It is created and only used by a GDA, it is read-only to users.

(g) `CDS_InCHName`

Indicates whether a directory or any of its descendants can store clearinghouse names. If this value is `true`, the directory can store clearinghouse names. If it is `false`, the directory cannot store clearinghouse names. This attribute is written by the system and is read-only to users. As of DCE 1.1 CDS will create this attribute on the cell root directory and give it a value of `true`, it will not appear in any other directory.[12]

(h) `CDS_LastSkulk`

Records the timestamp of the last skulk performed on this directory. This attribute is written by the system and is read-only to users.

(i) `CDS_LastUpdate`

Records the timestamp of the most recent change to any attribute of a directory replica, or any change to an entry in the replica. This attribute is written by the system and is read-only to users.

(j) `CDS_ObjectUUID`

Specifies the UUID of the directory, it is set by the system at directory creation time and, thereafter it is read-only, it may not be set by the user.

(k) `CDS_ParentPointer`

Contains a pointer to this directory's parent in the namespace. This attribute is written by the system and is read-only to users.

(l) `CDS_Replicas`

Specifies the address, UUID, and name of every clearinghouse where a copy of this directory is located. This attribute also specifies whether the replica in a particular clearinghouse is a master or read-only replica. This attribute is written by the system and is read-only to users.

(m) `CDS_ReplicaState`

Specifies whether a directory replica can be accessed. This attribute is written by the system and is read-only to users.

(n) `CDS_ReplicaType`

Indicates whether a directory replica is a master or read-only replica. This attribute is written by the system and is read-only to users.

(o) `CDS_ReplicaVersion`

Specifies the version of a replica of the directory. This attribute is written by the system and is read-only to users.

(p) `CDS_RingPointer`

Specifies the UUID of a clearinghouse containing another replica of this directory. This attribute is written by the system and is read-only to users. This attribute will appear on older directories but *not* on DCE 1.1 directories.

(q) `CDS_UpgradeTo`

A single-valued attribute used to control the upgrading of a directory from one version of CDS to another. By modifying this attribute, the process of upgrading a directory to a newer version of CDS may be initiated. After this attribute is set the background process notices it and tries to contact each replica. If it can then the `CDS_DirectoryVersion` attribute is changed to value of this attribute.[13]

(r) `CDS_UTS`

Specifies the timestamp of the most recent update to an attribute of the directory. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users. This

---

12. DCE 1.1 enforces that clearinghouses may only be created in a cell root directory. In previous versions this was just recommended.

13. This attribute was as not functional in pre-1.1 version of DCE.

attribute is written by the system and is read-only to users.

Supports the following operations:

(a) `directory add`

This operation creates a child pointer in the parent directory.

The argument is a list of names of parent directories to have child pointers added to them. The value of the required `-member` option is a list of names of child pointers to be added to each of the directories listed in the argument. Each child pointer name entered should only contain the last RDN of the name. The child object must exist or the command will return an error. The full name of a clearinghouse that holds a replica of the child directory is given as the value to the required `-clearinghouse` option. This option may only have one value and is used for each of the values of the `-member` option. Returns an empty string on success. If a child pointer of the same name already exists an error is returned.

This command is only needed to recreate a child pointer that was accidently deleted, i.e., in a troubleshooting situation. Normally child pointers are created internally by CDS when creating directories with the `directory create` command.

(b) `directory create`

Creates a new directory of the specified name. The argument is a list of names of directories to be created. Takes an optional `-attribute` option to specify a list of attributes to be included in each created directory. Added attributes are multi-valued unless a `-single` option is specified, in which case all attributes are single-valued. The `-single` option is only valid if the `-attribute` option is specified. Takes an optional `-clearinghouse` option to specify a clearinghouse to create all the directories in. The value of this option is not a list, it is only one clearinghouse name to create all the directories named in the argument in. If this option is not specified, the new directories are created in the same clearinghouse as the parent directory. Also takes a `-replica` option which indicates that a directory replica is created; when this option is used, the `-clearinghouse` option is required. Returns an empty string on success.

(c) `directory delete`

Used to delete a directory. When called without the `-tree` option, the directory must be empty or an error is generated. The argument is a list of names of directories to be deleted. There is a `-replica` and a `-clearinghouse` option which must be used together to delete a replica instead of a directory. The clearinghouse the replica is in is specified as the value of the `-clearinghouse` option; only one value can be specified, not a list.

An optional `-tree` option (takes no value) is supported to remove the named directory and everything (all directories, objects, links, and clearinghouses) beneath it. If the `-tree` option used, the `-replica` and `-clearinghouse` options are illegal. It is recommended that a `directory synchronize` (a.k.a. skulk) command be performed immediately before this operation is executed. Errors cause the operation to terminate immediately.

Returns an empty string on success. If a specified directory does not exist, an error is generated.

(d) `directory help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `directory list`

Returns a list of the names of all the descendants of a directory. The argument is a list of names of directories to be operated on. Descendants includes all directories, objects, links, and clearinghouses of the directory. Since only the names are returned there is no way to tell the class of each name unless by convention (e.g., most clearinghouses end with `_ch`). The following options can be used to specify the types of descendents to return: `-directories`, `-objects`, `-links`. They take no values and can be used in combination. By default fullnames are returned, the `-simplename` option can be used to return merely the last RDN of the name.

(f) `directory merge`

This operation copies the contents of one directory into another. The argument is the name of the source directory. This command takes a required `-into` option to specify the destination directory which must exist. For example, if `/.:/a` has two child objects `/.:/a/b` and `/.:/a/c`, then a `directory merge /.:/a -into /.:/x` would result (assuming no errors) in the following objects: `/.:/x/b` and `/.:/x/c`.[14]

Normally only the immediate contents of the directory are merged. This means all objects, links, and directories, but *not* the contents of child directories. To merge these as well, use the `-tree` option.

By default, the new objects are placed in the destination directory's clearinghouse, and all children (no matter how many level down) are placed in the same clearinghouse. To place any newly created descendent directories in another clearinghouse, use the `-clearinghouse` option with a value.[15] There can only be one clearinghouse specified for all directories involved in the merge operation. To specify more than one, either change this after the merge has happened, or use separate commands.

This command first checks for any collisions or ACL problems before beginning to merge any objects. If there are any problems encounted, an error is generated (not immediately, all objects are checked) and the names of all problem objects, links or directories are returned in a list. The administrator should then address these problems and rerun the merge command. If the `-nocheck` option is specified the check is not performed. This way time can be saved when trying a known non-problematic merge. This is *not* an atomic operation and other changes to the involved objects can cause problems. This command should be issued when others are *not* modifying the involved directories. Changing ACL's can be done to ensure this. If an error does occur during the actual merging process, it is generated and the operation aborts immediately.

The merge command actually re-creates the objects with the same writable attributes of the source objecs. This means that some read-only attributes will change between the source and destination. For example, the creation timestamp attribute (i.e., `CDS_CTS`) will change. Also, the ACL's of the destination objects will probably be different from the source objects. The ACL's on the destination objects are just those that are inherited from the containers initial container or initial object ACL's.[16]

(g) `directory modify`

Used to add, remove or change attributes and their values in a directory. The argument is a list of names of directories to be operated on. Attribute options are not supported; use one or more of `-add`, `-remove` or `-change` which each take an attribute list as an argument. All modifications are made to each directory listed in the argument. An error in any one causes the command to immediately abort and generate an error. The modifications are made (non-atomically) in the order specified on the command line. Returns an empty string on success.

Since attributes in CDS can be multi-valued, some ambiguity can result in the syntax of some commands. To resolve this ambiguity, this command takes two additional options that describe the values to the `-add` and `-remove` options. Multi-valued attributes are considered to be the default case, when dealing with

---

14. Note that these are *not*: `/.:/x/a/b` and `/.:/x/a/c`. This type of operation can be performed by first creating the target directory and then merging to that object. In this case:

```
dcecp> directory create /.:/x/a
dcecp> directory merge /.:/a -into /.:/x/a
```

15. Any existing directories will continue to reside in the clearinghouse they were in.

16. This was a topic of debate. Other models considered were as follows. Copy the source ACL to the destination object, but this requires that the user performing the operation have sufficient rights to create each object in the tree. A merge of the ACL's could be performed so that the inherited ACL is merged with the source ACL. This avoids the create rights problem, but the ACL on the destination objects are not obvious. The merge discussed usually involves converting `user` ACL entries to `foreign_user` ACL entries if the default cell of the two ACL's differ.

single-valued attributes the following options should be used:

(i) `-single`

If this option is present then the attributes specified in the `-add` option are to be single-valued. Normally all user defined attributes are defined to be multi-valued, even if only one value is specifed. This option is not legal without the `-add` option.

(ii) `-types`

If this option is present then the value of the `-remove` option is a list of attribute types, indicating that the entire attribute should be removed, not just a value. This option is not legal without the `-remove` option.

(h) `directory operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(i) `directory remove`

This operation deletes a child pointer from the directories specified. The argument is a list of names of directories to be operated on. The value of the required `-member` option is a list of child pointers (specified as only one RDN each) to be removed from each directory in the argument. Returns an empty string on success.

This command is only needed to delete a child pointer that accidently remains after the child directory is deleted. Normally child pointers are removed internally by CDS when deleting directories with the `directory delete` command.

(j) `directory show`

Returns a list of attributes in the directories specified. The argument is a list of names of directories to be operated on. If more than one directory is specified then all the arguments are grouped together in one list. The order of the returned arguments is the lexical order of the OID's of each attribute for each directory. For example:

```
dcecp> directory show /.:/hosts
{RPC_ClassVersion 0200}
{CDS_CTS 1994-06-18-20:16:52.888-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_UTS 1994-07-01-10:29:59.265-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_ObjectUUID 003d4d8e-d997-1db3-8259-0000c0f7de56}
{CDS_Replicas
 {{CH_UUID 0066ccea-d978-1db3-8259-0000c0f7de56}
  {CH_Name /.../terrapin/drkstr_ch}
  {Replica_Type Master}
  {Tower ncacn_ip_tcp 130.105.5.16}
  {Tower ncadg_ip_udp 130.105.5.16}}}
{CDS_AllUpTo 1994-07-01-20:30:18.316-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_Convergence medium}
{CDS_ParentPointer
 {{Parent_UUID 00146037-d97b-1db3-8259-0000c0f7de56}
  {Timeout
   {expiration 1994-05-02-23:29:13.749}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../terrapin/hosts}}}
{CDS_DirectoryVersion 3.0}
```

```
{CDS_LinkTimeout
 {expiration 1996-04-20-20:00:00.000}
 {extension +0-00:00:10.000I-----}}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1994-07-01-20:30:18.316-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_LastUpdate 1994-07-01-10:29:59.265-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_RingPointer 0066ccea-d978-1db3-8259-0000c0f7de56}
{CDS_Epoch 008fb182-d997-1db3-8259-0000c0f7de56}
{CDS_ReplicaVersion 3.0}
dcecp>
```

Takes a `-replica` and a `-clearinghouse` option to specify that the contents of a replica are to be returned. This will return the directory specific and replica specific attributes for the replica. The `-replica` option takes no value. The value of the `-clearinghouse` option is the name of one clearinghouse (not a list of names) to be used for all replicas. If one of the directories mentioned in the argument list is not in the specified clearinghouse then an error is returned.

Also takes an optional `-member` option with one required value which is the last RDN of child pointer in this directory. In this case the returned list describes the child pointer information for the specified member stored in the specified directories. This option may not be combined with the `-replica` or `-clearinghouse` option.

A `-schema` option can be used to return whether an attribute is single or multi-valued. This is specific to a directory, i.e., the same attribute can be single-valued on one directory and multi-valued on another directory. This option may not be used with other options. For example:

```
dcecp> directory show /.:/hosts -schema
{RPC_ClassVersion multi}
{CDS_CTS single}
{CDS_UTS single}
{CDS_ObjectUUID single}
{CDS_Replicas multi}
{CDS_AllUpTo single}
{CDS_Convergence single}
{CDS_ParentPointer multi}
{CDS_DirectoryVersion single}
{CDS_LinkTimeout single}
{CDS_ReplicaState single}
{CDS_ReplicaType single}
{CDS_LastSkulk single}
{CDS_LastUpdate single}
{CDS_RingPointer single}
{CDS_Epoch single}
{CDS_ReplicaVersion single}
dcecp>
```

(k) `directory synchronize`

Initiates a directory skulk of the directories specified. The argument is a list of names of directories to be operated on. Skulks are performed serially and begun immediately and the command does not return until all are finished. Returns an empty string on success.

8.11.7. link

The `link` object represents CDS softlinks. A softlink in the CDS directory service contains an attribute that has a name which is the name that the softlink points to. The softlink contains several built-in attributes, but users are not free to add their own. Softlinks can point to objects, directories, and other softlinks.

If the `_s(cds)` convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do *not* set the value of this variable after completion.

The following are the CDS defined attributes that may be present in a CDS softlink entry.

(a) `CDS_CTS`

Specifies the creation timestamp (CTS) of this softlink. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users.

(b) `CDS_LinkTarget`

Specifies the full name of the directory, object entry, or other softlink to which the softlink points.

(c) `CDS_LinkTimeout`

Specifies a timeout value after which the softlink is either renewed or deleted. Its value is a list of two elements:

(i) The expiration-time. A date and time after which CDS checks for the existence of the softlink's target and either extends or deletes the soft link. The value is specified as *yyyy-mm-dd-hh*`:`*mm*`:`*ss*, and portions of it can be defaulted.

(ii) The extension-time. A period of time by which to extend the softlink's expiration time (if the server has validated that the target still exists). The value is specified as *ddd-hh*`:`*mm*`:`*ss*, and portions of it can be defaulted.

(d) `CDS_UTS`

Specifies the timestamp of the most recent update to an attribute of the softlink. The value of this attributes is DTS style timestamp. This attribute is written by the system and is read-only to users.

Supports the following operations:

(a) `link create`

Creates a new softlink. The argument is a list of names of softlinks to be created. The `CDS_LinkTarget` attribute must be specified, either with the `-linkto` option, or with the `-attribute` option. The value is a single name that these new links point to. Addditionally the `CDS_LinkTimeout` attribute may be specified either with the `-attribute` option or with the `-timeout` option. The format for the value of these two options is slightly different. The following two commands do the same thing:

```
dcecp> link create /.:/foo -linkto /.:/bar -timeout {
> 1994-12-12-11:10:30.000 10-10:10:10}
dcecp> link create /.:/foo -attribute {
> {CDS_LinkTarget /.:/bar}
> {CDS_LinkTimeout
>   {expiration 1994-12-12-11:10:30.000}
>   {extension 10-10:10:10}}
> }
dcecp>
```

If the `CDS_LinkTimout` attribute is not included in the command line then the new softlinks are permanent and must be explicitly deleted. Returns an empty string on success.

(b) `link delete`

Deletes a softlink. The argument is a list of names of softlinks to be deleted. Returns an empty string on success. An error is returned if any of the objects do not exist.

(c) `link help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `link modify`

The `modify` operation can be used to change two attributes of a softlink: `CDS_LinkTarget` and `CDS_LinkTimeout`. The argument is a list of names of softlinks to be operated on. Takes the `-add`, `-remove`, and `-change` options to specify an attribute list to describe the changes. All the changes are performed on each softlink named in the argument. Returns an empty string on success.

(e) `link operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `link show`

Returns an attribute list of the attributes in the specified softlinks. The argument is a list of names of softlinks to be operated on. If more than one softlink is to be shown, the attributes of all the softlinks are concatenated into one list with a blank line between softlinks. The order of the returned attributes is the lexical order of the OID's of each attribute for each softlink. For example:

```
dcecp> link show /.:/link
{CDS_CTS 1994-07-01-22:11:52.519-05:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-07-01-22:11:53.069-05:00I0.000/00-00-c0-f7-de-56}
{CDS_LinkTarget /.../terrapin/hosts}
{CDS_LinkTimeout
 {expiration 1996-05-01-20:00:00.000}
 {extension +0-10:10:10.000I-----}}
```

8.11.8. clearinghouse

The `clearinghouse` object represents CDS clearinghouses. Clearinghouses are databases located on CDS server machines that store the data in the CDS directory service. On the server machines are files that contain the actual clearinghouse data. Clearinghouses are also represented in the CDS namespace by an entry that contains information about the clearinghouse.

If the `_s(cds)` convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do *not* set the value of this variable after completion.

The following are the CDS defined attributes that may be present in a clearinghouse:

(a) `CDS_AllUpTo`

Indicates the date and time the clearinghouse object has been updated to reflect the `CDS_CHDirectories` attribute.

(b) `CDS_CHDirectories`

Specifies the full name and unique identifier (UUID) of every directory that has a replica in this clearinghouse.

(c) `CDS_CHLastAddress`

Specifies the current reported network address of the clearinghouse.

(d) `CDS_CHName`

Specifies the full name of the clearinghouse.

(e) `CDS_CHState`

Specifies the state of the clearinghouse. The state `on` indicates the clearinghouse is running and available.

(f) `CDS_CTS`

Specifies the creation timestamp (CTS) of the clearinghouse.

(g) `CDS_DirectoryVersion`

Specifies the current version of the directory in the clearinghouse in which the directory was created.

(h) `CDS_NSCellname`

Specifies the name of the cell in which the clearinghouse resides.

(i) `CDS_ObjectUUID`

Specifies the UUID of the clearinghouse, it is set by the system at clearinghouse creation time and, thereafter it is read-only, it may not be set by the user.

(j) `CDS_ReplicaVersion`

Specifies the current version of the replica in which the directory was created.

(k) `CDS_UpgradeTo`

A single-valued attribute used to control the upgrading of a directory from one version of CDS to another. By modifying this attribute, the process of upgrading a directory to a newer version of CDS may be initiated.

(l) `CDS_UTS`

Specifies the timestamp of the most recent update to an attribute of the clearinghouse.

The counters are:

(a) `badchs`

Specifies the number of times the `clearinghouse entry missing` event was generated.

(b) `baddata`

Specifies the number of times that the `data corruption` event was generated.

(c) `badskulks`

Specifies the number of times that a skulk of a directory, initiated from this clearinghouse, failed to complete— usually because one of the replicas in the replica set was unreachable.

(d) `badupgrades`

Specifies the number of times that upgrades failed.

(e) `disables`

Specifies the number of times that the clearinghouse was disabled since it was last started.

(f) `enables`

Specifies the number of times that the clearinghouse was enabled since it was last started.

(g) `partials`

Specifies the number of requests directed to this clearinghouse that resulted in the return of a partial answer instead of satisfying the client's request.

(h) `reads`

Specifies the number of read operations directed to this clearinghouse.

(i) `rootlosts`

Specifies the number of times the `root lost` event was generated.

(j) `writes`

Specifies the number of write operations directed to this clearinghouse.

Supports the following operations:

(a) `clearinghouse catalog`

Returns a list of the names of all clearinghouses in the specified cell. The optional argument specifies the cell and defaults to `/.:`.

(b) `clearinghouse create`

Creates a new clearinghouse on the local machine. Takes one argument which is the name of a new clearinghouse (it is not a list). Clearinghouses should only be named in the root directory (i.e., `/.:`). When this operation is invoked, a read-only replica of the root directory is stored in the new clearinghouse. Because the process that creates the new clearinghouse initiates a skulk of the root directory, all replicas of the root should be reachable when you enter the command. To ensure this, perform an immediate skulk of `/.:` prior to invoking this command, using the `directory synchronize /.:` command. Returns an empty string on success.

(c) `clearinghouse delete`

This operation deletes the clearinghouse given as an argument from the *local* machine. The argument is a list of names of clearinghouses to be deleted. Clearinghouses that contain master replicas of directories may not be deleted. This command also automatically deletes all read-only replicas from the clearinghouse, however it is recommended that all read-only replicas be deleted by hand *before* invoking this command since invoking many skulks will cause the command to execute slower. Returns an empty string on success.

(d) `clearinghouse disable`

This operation removes knowledge of the specified clearinghouse from the server running on the *local* machine. The argument is a list of names of clearinghouses to be operated on. It is useful when relocating a clearinghouse. Specifically the name of the prefix of the clearinghouse files is removed from the `/opt/dcelocal/var/directory/cds/cds_files` file and the local server is contacted and notified that the clearinghouse is disabled. The clearinghouse entry is *not* removed from the namespace, nor are the datafiles associated with the clearinghouse removed. Returns an empty string on success.

(e) `clearinghouse help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `clearinghouse initiate`

Triggers an immediate checkpoint on the specified clearinghouse. The argument is a list of names of clearinghouses. There is a required `-checkpoint` option. Requires write permission to the CDS server. Returns an empty string on success.

(g) `clearinghouse operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `clearinghouse repair`

This command will repair a clearinghouse in a broken state. In DCE 1.1 it can only repair bad timestamps, one where the `CDS_UTC` value is in the future. Requires the `-timestamps` option. This command will disable the clearinghouse, scan for bad timestamps and correct them, checkpoint and re-enable the clearinghouse. The argument is a list of names of clearinghouses. Requires write permission to the CDS server.

Returns an empty string on success.

(i) `clearinghouse show`

Returns an attribute list showing the attributes of the clearinghouse named as the argument. The argument is a list of names of clearinghouses to be operated on. If more than one clearinghouse is given as an argument, all the attributes from each are concatenated together into one list with a blank line between clearinghouses. The order of the returned attributes is the lexical order of the OID's of each attribute for each clearinghouse. For example:

```
dcecp> clearinghouse show /.:/drkstr_ch
{CDS_CTS 1994-06-18-20:16:22.150-05:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-06-19-17:17:43.911-05:00I0.000/00-00-c0-f7-de-56}
{CDS_ObjectUUID 0066ccea-d978-1db3-8259-0000c0f7de56}
{CDS_AllUpTo 1994-07-01-21:30:18.948-05:00I0.000/\
00-00-c0-f7-de-56}
{CDS_DirectoryVersion 3.0}
{CDS_CHName /.../terrapin/drkstr_ch}
{CDS_CHLastAddress
 {Tower ncacn_ip_tcp 130.105.5.16}
 {Tower ncadg_ip_udp 130.105.5.16}}
{CDS_CHState on}
{CDS_CHDirectories
 {{Dir_UUID 00146037-d97b-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin}}
 {{Dir_UUID 0043797a-d991-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys}}
 {{Dir_UUID 004faa42-d992-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/HP}}
 {{Dir_UUID 004fa65a-d993-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/HP/sample-apps}}
 {{Dir_UUID 004b1130-d994-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/dce}}
 {{Dir_UUID 00498a0e-d995-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/dce/sec}}
 {{Dir_UUID 003ed80c-d996-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/dce/dfs}}
 {{Dir_UUID 003d4d8e-d997-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/hosts}}
 {{Dir_UUID 003bc522-d998-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/hosts/drkstr}}
 {{Dir_UUID 0089ee8c-44e0-1dbe-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/help}}
 {{Dir_UUID 001c6cea-00fb-1dc5-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/test_1}}
 {{Dir_UUID 00440fe8-02a1-1dc5-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/dirmod}}}
{CDS_ReplicaVersion 3.0}
{CDS_NSCellname /.../terrapin}
dcecp>
```

If given a `-counters` option returns an attribute list of the counters in the clearinghouse. The `-all` option can be used to return both attribute and counter information. The order is the attributes first then the counters as if the two commands were done one after the other. Counters are displayed in the order returned by the server, for example:

```
dcecp> clearinghouse show /.:/foo_ch -counters
{corruptions 0}
{disables 0}
{enables 1}
{failedupgrades 0}
{missingentries 0}
{reads 2336}
{returnedrefs 2}
{rootunreachables 0}
{skulkfailures 0}
{writes 68}
```

(j) `clearinghouse verify`

Insures that a clearinghouse is in a correct state by performing the following checks. Requires write permission to the CDS server. Returns an empty string on success.

(i) Skulks directories if it is time to do so based on the value of the `CDS_Convergence` attribute.

(ii) Upgrades directory replicas if a `CDS_UpgradeTo` attribute indicates to do so.

(iii) Ensures that child and parent pointer information is correct. This involves ensuring that the set of replicas stored in the child pointer matches the set of replicas stored in the actual directory.

(iv) Performs any need IP address changes in directories.

(v) Checks for primary cellname changes in the clearinghouse.

(vi) Removes all old data (replicas and attributes) based on the value of the `CDS_AllUpTo` attribute.

8.11.9. cdscache

The `cdscache` object represents the cds cache on the local node. The CDS cache contains information about servers and clearinghouses known to the local machine, and also contains user data about CDS entries that have been read. The `create` and `delete` operations only apply to the server information. The `show` and `dump` commands can display other information. Supports the following operations:

(a) `cdscache create`

This operation adds information about a CDS server to the local CDS cache. The argument is a list of names of servers to be added to the cache. These are the simple names of the machine, not the name of entry in the DCE namespace for the server (e.g. `absolut` not `/.:/hosts/absolut/self`). There is also one required option, `-binding`. This is the binding information for the server to be added to the CDS cache. The value must be a single partial string binding, not a list. Returns an empty string on success.

(b) `cdscache delete`

This operation is used to remove information about a CDS server from the local CDS cache. The argument is a list of names of servers to be deleted from the cache. It can only be used to remove information added with the `cdscache create` command. Returns an empty string on success.

(c) `cdscache dump`

Returns all information stored in the local CDS cache. It takes no argument. Returns an empty string on success. This command is meant to be used when debugging CDS problems. As such the output is not in a form easily parsed by Tcl. The user must have root privileges to run this command. This command may be removed from a production environment.

(d) `cdscache help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `cdscache operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `cdscache show`

Returns information stored in the CDS cache. The argument is a list of names of either clearinghouses or (simple names of) servers about whose information from the cache is returned. Must be invoked with only one of the following options: `-server` or `-clearinghouse` to identify the type of the argument.

```
dcecp> cdscache show /.:/drkstr_ch -clearinghouse
{CH_Name /.../terrapin_cell.osf.org/drkstr_ch}
{Created 1994-07-19-15:56:54.997}
{Others 1512}
{Reads 3467}
{Tower {ncacn_ip_tcp 130.105.5.16}}
{Tower {ncadg_ip_udp 130.105.5.16}}
{Writes 249}
dcecp> cdscache show drkstr -server
{CH_Name /.../terrapin_cell.osf.org/drkstr_ch}
{Tower {ncacn_ip_tcp 130.105.5.16}}
{Tower {ncadg_ip_udp 130.105.5.16}}
```

8.11.10. principal

The `principal` object represents Registry principals. Unless otherwise noted, all of the operations of this object take one argument which is a list of names of principals to act on. They must be principal names (i.e., names relevant to the principal domain of the form `/.../`*`cellname`*`/`*`princname`* or *`princname`* to default the cellname.), not the names of the database objects that contains registry information about principals (i.e., they should not contain the string `/sec/principal/`). The names may be fully qualified or not, but if any are not fully qualified, then none may be.[17] Non-fully qualified names refer to a principal in the cell described in `_s(sec)` or in the cell of the local host if `_s(sec)` is not set.

After this command executes, the `_b(sec)` convenience variable is set to the name of the server that was bound to for the command. The value of the variable `_s(sec)` before the command is treated as a hint, i.e., the server specified will be contacted if it can service the request. A case where it can't service the request is if a read-only registry was bound to, and the next command is a `create` command. In this case, the master registry will be bound to automatically and the `_b(sec)` variable updated appropriately. The value of the variable is the name of registry bound to in one of the formats specified as valid for the argument to the `registry` object.

In the registry, each principal has a primary name, a value for each of the following attributes, and possibly some extended registry attributes:

(a) `alias`

The value of this attribute is either `yes` or `no`. Each principal can have only one name, but may have one or more alias names. All these names refer to the same principal, and therefore the same UUID and uid. While aliases refer to the same principal they are separate entries in the registry database. Therefore the instance name given to a `principal` command can refer to either the primary name or an alias name of a principal. The value of this attribute determines this.

---

17. This limitation is to prevent possible ambiguity in the following case. Assume `/.:` refers to cell `/.../foo`.

```
dcecp> set _s(sec) /.../bar
dcecp> princ create {/.../zot/tom dick harry}
```

What cell did the user intend to place `dick` and `harry` in?

(b) `fullname`

The full name of the principal, it is for information purposes only. It typically describes or expands a primary name to allow easy recognition by users. For example, a principal could have a primary name of `jsbach` and a fullname of `Johann S. Bach`. The value is a string, if it contains spaces, it is displayed in quotes, and on entry must be in quotes or braces (as per Tcl quoting rules). If not entered, the fullname defaults to the null string (that is, blank).

(c) `quota`

Specifies the principal's object creation quota, the total number of registry objects that can be created by the principal. It is either a non-negative number or the string `unlimited`. A value of `0` prohibits the principal from creating any registry objects. Each time a principal creates a registry object, this value is decremented for that principal.

(d) `uid`

The User Identifier for the principal. No two principals can have the same uid. However, aliases can share one uid. It is often called the Unix ID and is an integer.

(e) `uuid`

An internal identifier for a principal. No two principals can have the same UUID.

Supports the following operations:

(a) `principal catalog`

Returns a list of the names of all principals in the registry. It takes no arguments. By default, fully qualified names are returned in the form *`cellname/principalname`*. If the `-simplename` option is given, then the cellname is not prepended to the front of the principal names. Names are returned in lexical order.

(b) `principal create`

Creates a new principal. The argument is a list of names of principals to be created. Returns an empty string on success. Options are used to specify the attributes of the newly created principal. All options are applied to all principals in the argument.[18]

Accepts a `-attribute` option to specify the attributes as an attribute list. The attribute options are:

(i) `-alias`

Takes a value of either `yes` or `no` to specify if the newly created principal is an alias or not. The value of this attribute defaults to `no`.[19] When creating an alias, the `-uid` option is required to specify the uid of the primary principal to alias. For example, the following command creates an alias called `postmaster` for the principal with uid 1234:

```
principal create postmaster -uid 1234 -alias yes
```

(ii) `-fullname`

The full name of the principal to be added to the registry. The value is a string. If it contains spaces, it must be in quotes or braces. If not entered, the fullname defaults to the null string (that is, blank).

(iii) `-quota`

18. Therefore do not use the `-uid` option when creating two principals with the same command, the second will fail since the uid is already in use after the first is created. However, the `-alias` and `-uid` options can be used to create several aliases to an existing principal with one command.

19. So specifying `-alias no` is never required.

Specifies the principal's object creation quota, the total number of registry objects that can be created by the principal. It is either a non-negative number or the string `unlimited`. If you do not specify this option the object creation quota defaults to `unlimited`.

(iv) `-uid`

The User Identifier for the principal. No two principals can have the same uid, however, aliases can share one uid with a principal. If this option is not present, then a uid will be assigned to the principal automatically. The value of this option is an integer.

(v) `-uuid`

This option is only used to ''adopt'' an orphaned UUID. The value of this option is a UUID. Normally the UUID for a new principal is generated by the registry. In cases where data exists tagged with a UUID of a principal that has been deleted from the registry, this option can be used on the `create` command to specify the old UUID for a new principal. The UUID specified *must* be an orphan, i.e., a UUID for which no name exists in the registry. An error occurs if you specify a name (or uid, see below) that is already defined in the registry.

The `-alias` option may *not* be used with this option. Both the `-fullname` and `-quota` options may be.

If the principal is a cell principal, the `-uid` option can be used to specify the UNIX number to be associated with the principal. (If you do not enter this option for a cell principal, the next sequential UNIX number is supplied as a default by the registry.) For all principals other than cell principals, the uid is extracted from information embedded in the principal's UUID and cannot be specified here.

(c) `principal delete`

This operation deletes principals from the registry. When a principal is deleted, the principal's account is deleted as well. The argument is a list of names of principals to be deleted. Note that these names can be either a primary or alias names. In either case, an account associated with that name is deleted. If a named principal does not exist an error is generated. Returns an empty string on success.

(d) `principal help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `principal modify`

This operation is used to change attributes of principals. The argument is a list of names of principals to be operated on. All modifications are applied to all principals named in the argument. Principals are modified in the order they are listed and all modifications to an *individual* principal are atomic. Modifications to multiple principals are not atomic. A failure for any one principal in a list causes an error to be generated and the rest of the operation to be aborted. Returns an empty string on success.

The `-change` option can be used to modify the value of any one of the above listed attributes (excep `uid` and `uuid`) or any **ERA**. The value of the `-change` option is an attribute list describing the new values for the specified attributes. Also supports the following attribute options: `-alias`, `-quota`, and `-fullname`. The `-add` and `-remove` options can be used to specify any ERA's, but not the above listed attributes, as each of them must always have a value.

(f) `principal operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `principal rename`

This operation changes the name of a specified principal. The argument is a single name of a principal to be renamed. Takes a required `-to` option with a value of the new name. The value may not be a list. Returns an empty string on success.

(h) `principal show`

This operation returns an attribute list describing the specified principals. The argument is a list of names of principals to be operated on. If more than one principal is given, then the attributes are concatenated together with a blank line between principals. There is one attribute in addition to the attributes described above. It is called `groups` and its value is a list of group names that the principal is a member of. Attributes are returned in the following order: `fullname`, `uid`, `uuid`, `alias` and `quota` followed by `groups`. If called with the `-xattrs` option then Extended Registry Attributes are returned instead of the above attributes. If called with `-all` then both are returned.

8.11.11. group

The `group` object represents Registry groups. Groups are collections of principal names. Unless otherwise noted, all of the operations of this object take one argument which is the name of the group to act on. It must be a group name, not the name of the database object that contains the registry information about the group (i.e., it should not begin with `/.:/sec/group/`). The names may be fully qualified or not, but if any are not fully qualified, then none may be. None fully qualified names refer to a group in the cell described in `_s(sec)` or in the cell of the local host if `_s(sec)` is not set. See the principal section for a description of how the `_s(sec)` and `_b(sec)` variables are handled by this command.

In the registry, each group has a primary name, a value for each of the following attributes, and possibly some extended registry attributes:

(a) `alias`

The value of this attribute is either `yes` or `no`. Each group can have only one name, but may have one or more alias names. All these names refer to the same group, and therefore the same UUID and gid. While aliases refer to the same group they are separate entries in the registry database. Therefore the instance name given to a `group` command can refer to either the primary name or an alias name of a group. The value of this attribute determines this.

(b) `fullname`

The full name of the group, it is for information purposes only. It typically describes or expands a primary name to allow easy recognition by users. For example, a group could have a primary name of `dce` and a fullname of `DCE Development`. The value is a string, if it contains spaces, it is displayed in quotes, and on entry must be in quotes or braces (as per Tcl quoting rules). If not entered, the fullname defaults to the null string (that is, blank).

(c) `gid`

The User Identifier for the group. No two groups can have the same gid, however, aliases can share one gid. It is often called the Unix ID and is an integer.

(d) `inprojlist`

The value of this attribute is either `yes` or `no`. If it is `no` then members of this group will not accrue the access rights of this group.

(e) `uuid`

An internal identifier for a group. No two groups can have the same UUID.

Supports the following operations:

(a) `group add`

This command is used to add a members to a group. The argument is a list of names of groups to be have members added to. The value of the required `-member` option is a list of names of principals to be added to each group in the argument. The principals must exist or the command will return an error. Returns an empty string on success.

(b) `group catalog`

Returns a list of the names of all groups in the registry. It takes no arguments. By default, fully qualified names are returned in the form *`cellname/groupname`*. If the `-simplename` option is given then the cellname is not prepended to the front of the group names. The groups are returned in lexical order.

(c) `group create`

Creates a new group. The argument is a list of names of groups to be created. Returns an empty string on success. Options are used to specify the attributes of the newly created group. All options are applied to all groups in the argument.[20]

Accepts a `-attribute` option to specify the attributes as an attribute list. The attribute options are:

(i) `-alias`

Takes a value of either `yes` or `no` to specify if the newly created group is an alias or not. The value of this attribute defaults to `no`.[21] When creating an alias, the `-gid` option is required to specify the gid of the primary group to alias.

(ii) `-fullname`

The full name of the group to be added to the registry. The value is a string. If it contains spaces, it must be in quotes or braces. If not entered, the fullname defaults to the null string (that is, blank).

(iii) `-gid`

The Group Identifier for the group. If this option is not present, then a gid will be assigned to the group automatically.

(iv) `-inprojlist`

Specifies if the group is to be included in the principal's project list. The value of this field is either `yes` or `no`. If this option is not included in the command, then the `yes` value is used as a default.

(v) `-uuid`

This option is only used to ''adopt'' an orphaned UUID. Normally the UUID for a new group is generated by the registry. In cases where data exists tagged with a UUID of a group that has been deleted from the registry, this option can be used on the `create` command to specify the old UUID for a new group. The UUID specified *must* be an orphan, i.e., a UUID for which no name exists in the registry. An error occurs if you specify a name that is already defined in the registry.

Neither the `-alias` or `-gid` options may be used with this option. Both the `-fullname` and `-inprojlist` options may be.

(d) `group delete`

This operation deletes a group from the registry. When a group is deleted, any accounts associated with that group are deleted as well. The argument is a list of names of groups to be deleted. If the groups do not exist an error is generated. Returns an empty string on success.

(e) `group help`

---

20. Therefore do not use the `-gid` option when creating two groups with the same command, the second will fail since the gid is already in use after the first is created. However, the `-alias` and `-gid` options can be used to create several aliases to an existing group with one command.

21. So specifying `-alias no` is never required.

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `group list`

Returns a list of the names of all members of a group. The argument is a list of names of groups to be operated on. If more than one group is listed, the names are concatenated together with a blank line between groups. By default, fully qualified names are returned in the form *`cellname/name`*. If the `-simplename` option is given then the cellname is not prepended to the front of the member names. The members of each group are listed in lexical order. Returns an empty string on success.

(g) `group modify`

This operation is used to change attributes of groups. The argument is a list of names of groups to be operated on. All modifications are applied to all groups named in the argument. Groups are modified in the order they are listed and all modifications to an *individual* group are atomic. Modifications to multiple groups are not atomic. A failure for any one group in a list causes an error to be generated and the rest of the operation to be aborted. Returns an empty string on success.

The `-change` option can be used to modify the value of any one of the above listed attributes (excep `gid` and `uuid`) or any ERA. The value of the `-change` option is an attribute list describing the new values for the specified attributes. Also supports the following attribute options: `-alias`, `-inprojlist`, and `-fullname`. The `-add` and `-remove` options can be used to specify any ERA's, but not the above listed attributes, as each of them must always have a value.

(h) `group operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(i) `group remove`

This command is used to remove a member from a group. The argument is a list of names of groups to have members removed from. The value of the required `-member` option is a list of names of principals which will be removed from the groups listed in the argument. When a member is removed from a group, any accounts associated with that principal and group are deleted.[22] Returns an empty string on success.

(j) `group rename`

This operation changes the name of a specified group. The argument is a single name of a principal to be renamed. Takes a required `-to` option with a value of the new name. The value may not be a list. Returns an empty string on success.

(k) `group show`

This operation returns an attribute list describing the specified groups. The argument is a list of names of groups to be operated on. If more than one group is given, then the attributes are concatenated together with a blank line between groups. Attributes are returned in the following order: `fullname`, `gid`, `uuid`, `alias` and `inprojlist`. If called with the `-xattrs` option then Extended Registry Attributes are returned instead of the above attributes. If called with `-all` then both are returned.

8.11.12. organization

The `organization` object represents Registry organizations. Organizations are collections of principal names. Unless otherwise noted, all of the operations of this object take one argument which is the name of the organization

---

22. Remember that accounts are associated with a principal, a group, and an organization; so the statement means that any accounts where *both* the principal name and group name match those given to this command are removed, but accounts where only one matches are untouched.

to act on. It must be an organization name, not the name of the database object that contains the registry information about the organization (i.e., it should not begin with `/.:/sec/org/`). The names may be fully qualified or not, but if any are not fully qualified, then none may be. None fully qualified names refer to a organization in the cell described in `_s(sec)` or in the cell of the local host if `_s(sec)` is not set. See the principal section for a description of how the `_s(sec)` and `_b(sec)` variables are handled by this command.

Each organization has both attributes and policies associated with them. Policies regulate such things as account and password lifetimes for all accounts associated with a particular organization. In fact, these policies are the main feature that organizations provide. There are two types of policies referred to here, those associated with a particular organization, and those that are registry wide. Please note that there are other policies that are associated with accounts (and they also can be registry-wide).

In the registry, each organization has a primary name, a value for each of the following attributes, and possibly some extended registry attributes:

(a) `fullname`

The full name of the organization, it is for information purposes only. It typically describes or expands a primary name to allow easy recognition by users. For example, an organization could have a primary name of `osf` and a fullname of `Open Software Foundation`. The value is a string, if it contains spaces, it is displayed in quotes, and on entry must be in quotes or braces (as per Tcl quoting rules). If not entered, the fullname defaults to the null string (that is, blank).

(b) `orgid`

The User Identifier for the organization. No two organizations can have the same orgid. However, aliases can share one orgid. It is often called the Unix ID and is an integer.

(c) `uuid`

An internal identifier for a organization. No two organizations can have the same UUID. In normal use, this attribute will be set by the system and will be displayed for administrators, in some unusual cases an admin might want to set this value explicitly.

The policies are:

(a) `acctlife`

This policy defines the lifespan of accounts. It is specified as either an integer (for which the units are days) or the string `unlimited`.

(b) `pwdalpha`

This policy defines whether or not passwords can consist entirely of alphanumeric characters. Its value is either `yes` or `no`.

(c) `pwdexpdate`

This policy defines a date on which a password will expire. The date is entered as an internationalized date string or the string `none` in which case, there is no expiration date for the password.

(d) `pwdlife`

This policy defines the lifespan of passwords. It is specified as either an integer number of days or the string `unlimited`. The tag `days` can follow the integer on output and is ignored on input, for example:

```
{pwdlife 22 days}
```

(e) `pwdminlen`

This policy defines the minimum number of characters in a password. Its value is a positive integer or the integer `0` which means there is no minimum length.

(f) `pwdspaces`

This policy defines whether or not passwords can consist entirely of spaces. Its value is either `yes` or `no`.

Supports the following operations:

(a) `organization add`

This command is used to add a members to an organization. The argument is a list of names of organizations to be have members added to. The value of the required `-member` option is a list of names of principals to be added to each organization in the argument. The principals must exist or the command will return an error. Returns an empty string on success.

(b) `organization catalog`

Returns a list of the names of all organizations in the registry. It takes no arguments. By default, fully qualified names are returned in the form *`cellname/name`*. If the `-simplename` option is given then the cellname is not prepended to the front of the organization names. The organizations are returned in lexical order.

(c) `organization create`

Creates a new organization. The argument is a list of names of organizations to be created. Returns an empty string on success. Options are used to specify the attributes of the newly created organization. All options are applied to all organizations in the argument.[23]

Supports both attribute lists (with a `-attribute` option) and attribute options named after each attribute and policy. If an orgid is not entered, then one will be assigned to the organization automatically. A UUID should only be specified to ''adopt'' an orphaned UUID. Normally the UUID for a new organization is generated by the registry. In cases where data exists tagged with a UUID of an organization that has been deleted from the registry, it can be entered on the `create` command line to specify the old UUID for a new organization. The UUID specified *must* be an orphan, i.e., a UUID for which no name exists in the registry. An error occurs if you specify a name that is already defined in the registry. The `orgid` attribute may *not* be specified if the `uuid` is, but the `fullname` attribute may be. Examples:

```
org create dce -attr {{fullname {Dist Comp Env}} \
{orgid 101}}
org create dce -fullname {Dist Comp Env} -orgid 101
org create dce -fullname {Dist Comp Env} \
-uuid c2aac790-dc6c-11cc-a6f8-080009251352
```

(d) `organization delete`

This operation deletes the organization from the registry. When an organization is deleted, any accounts associated with that organization are deleted as well. The argument is a list of names of organizations to be deleted. If the organizations do not exist an error is generated. Returns an empty string on success.

(e) `organization help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `organization list`

Returns a list of the names of all members of a organization. The argument is a list of names of organizations to be operated on. If more than one organization is listed, the names are concatenated together with a blank line between organizations. By default, fully qualified names are returned in the form *`cellname/name`*. If the `-simplename` option is given then the cellname is not prepended to the front

23. Therefore do not use the `-orgid` option when creating two organizations with the same command, the second will fail since the orgid is already in use after the first is created. However, the `-alias` and `-orgid` options can be used to create several aliases to an existing organization with one command.

of the member names. The members of each organization are listed in lexical order. Returns an empty string on success.

(g) `organization modify`

This operation is used to change attributes and policies of organizations. To change registry-wide policies, see the `registry` command. The argument is a list of names of organizations to be operated on. All modifications are applied to all organizations named in the argument. Organizations are modified in the order they are listed and all modifications to an *individual* organization are atomic. Modifications to multiple organizations are not atomic. A failure for any one organization in a list causes an error to be generated and the rest of the operation to be aborted. Returns an empty string on success.

The `-change` option can be used to modify the value of any one of the above listed attributes (except `orgid` and `uuid`) or any ERA. The value of the `-change` option is an attribute list describing the new values for the specified attributes. Also supports the following attribute options: `-fullname`, `-acctlife`, `-pwdalpha`, `-pwdexpdate`, `-pwdlife`, `-pwdminlen`, and `-pwdspaces`. Examples:

```
org modify dce -attr {{fullname {Dist Comp Env}} \
{orgid 101}}
org modify dce -fullname {Dist Comp Env} -orgid 101
```

The `-add` and `-remove` options can be used to specify any ERA's, but not the above listed attributes, as each of them must always have a value.

(h) `organization operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(i) `organization remove`

This command is used to remove a member from a organization. The argument is a list of names of organizations to have members removed from. The value of the required `-member` option is a list of names of principals which will be removed from the organizations listed in the argument. When a member is removed from a organization, any accounts associated with that principal and organization are deleted.[24] Returns an empty string on success.

(j) `organization rename`

This operation changes the name of a specified organization. The argument is a single name of a principal to be renamed. Takes a required `-to` option with a value of the new name. The value may not be a list. Returns an empty string on success.

(k) `organization show`

This operation returns an attribute list describing the specified organizations. The argument is a list of names of organizations to be operated on. If more than one organization is given, then the attributes and policies are concatenated together with a blank line between organizations. Accepts a `-policies` option that returns the polices of the organization instead of the attributes. If called with the `-xattrs` option then Extended Registry Attributes are returned instead of the above attributes. A `-all` option can be used to return the attributes followed by ERA's and the policies.

24. Remember that accounts are associated with a principal, a group, and an organization; so the statement means that any accounts where *both* the principal name and organization name match those given to this command are removed, but accounts where only one matches are untouched.

Attributes are returned in the following order: `fullname`, `orgid`, `uuid`. Policies are returned in the following order: `acctlife`, `pwdalpha`, `pwdexpdate`, `pwdlife`, `pwdminlen`, and `pwdspaces`. If the organization does not have any policies then `nopolicies` is returned.

The policies that are actually in effect can be different from the organization policies due to conflicts with registry wide policies. If this is the case, the show command will alter the attribute structure on output to include an `effective` tag and the effective value, much in the same way ACL's handle masks (see *ACL's* in the *Data Structures* section for details). For example:

```
dcecp> org show foo -policies
{acctlife 30 days}
{pwdalpha no}
{pwdexpdate none}
{pwdlife unlimited effective 5 days}
{pwdminlen 6}
{pwdspaces no}
```

8.11.13. account

The `account` object represents Registry accounts. Each account is associated with one principal, one group and one organization. However, accounts are named by just the principal name. Aliases are differentiated for principals, so one principal can have multiple accounts under different alias names. The account has attributes to specify the group and organization of the account, these must be specified on account creation.

Unless otherwise noted, all of the operations of this object take one argument which is a list of names of accounts (principals) to act on. They must be principal names (i.e., names relevant to the principal domain of the form `/.../`*`cellname`*`/`*`princname`* or *`princname`* to default the cellname.), not the names of the database objects that contains registry information about principals (i.e., they should not contain the string `/sec/principal/`). The names may be fully qualified or not, but if any are not fully qualified, then none may be. None fully qualified names refer to a principal in the cell described in `_s(sec)` or in the cell of the local host if `_s(sec)` is not set. See the principal section for a description of how the `_s(sec)` and `_b(sec)` variables are handled by this command.

Accounts contain the following attributes:

(a) `acctvalid`

A flag set to determine account validity. Possible values are either `yes` or `no`. An account with an `acctvalid` attribute set to `no` is invalid and cannot be logged into. The default is `yes`.

(b) `client`

A flag set to indicate whether or not the account is for a principal that can act as a client. Possible values are either `yes` or `no`. If you set this flag to `yes`, the principal is able to log in to the account and acquire tickets for authentication. The default is `yes`.

(c) `created`

A list of two items. The first is the principal name of the creator of the account, the second is an ISO timestamp showing the time of creation. This attribute is set by the system and may not be modified.

(d) `description`

A text string (limited to PCS) typically used to describe the use of the account.

(e) `dupkey`

A flag set to determine if tickets issued to the account's principal can have duplicate keys. Possible values are either `yes` or `no`. The default is `no`.

(f) `expdate`

The date on which the account expires. To renew the account, change the date in this field. The value is an ISO timestamp or the string `none.`. The default is `none`.

(g) `forwardabletkt`

A flag set to determine whether a new ticket-granting ticket with a network address that differs from the present ticket-granting ticket network address can be issued to the account's principal. The `proxiabletkt` attribute performs the same function for service tickets. Possible values are either `yes` or `no`. The default is `yes`.

(h) `goodsince`

The date and time the account was last known to be in an uncompromised state. Any tickets granted before this date are invalid. The value is an ISO timestamp. When the account is initially created, the `goodsince` date is set to the current date.

Control over this date is especially useful if you know that an account's password was compromised. Changing the password can prevent the unauthorized principal from accessing the system again using that password, but it does not prevent the principal that is already logged in. There is no way to cancel an already issued arbitrary ticket (you must wait for it to timeout), but you can cancel a previously issued TGT. If you set the `goodsince` date to the date and time the compromised password was changed, all TGT's issued before that time become invalid and the unauthorized principal will not be able to obtain new tickets.

(i) `group`

The name of the group associated with the account. The value is a single group name of an existing group in the registry. This attribute must be specified on an `account create` command, it does not have a default value. If this group is deleted from the registry, then the account is as well.

(j) `home`

The filesystem directory in which the principal is placed in at login.

(k) `lastchange`

A list of two items. The first is the principal name of the last modifier of the account, the second is an ISO timestamp showing the time of the last modification. This attribute is set by the system and may not be modified directly.

(l) `organization`

The name of the organization associated with the account. The value is a single organization name of an existing organization in the registry. This attribute must be specified on an `account create` command, it does not have a default value. If this organization is deleted from the registry, then the account is as well.

(m) `postdatedtkt`

A flag set to determine if tickets with a start time some time in the future can be issued to the account's principal. Possible values are either `yes` or `no`. The default is `no`.

(n) `proxiabletkt`

A flag set to determine whether or not a new ticket with a different network address than the present ticket can be issued to the account's principal. The `forwardabletkt` attribute performs the same function for ticket-granting tickets. Possible values are either `yes` or `no`. The default is `no`.

(o) `password`

The value of this attribute is the password of the account. There is no default value, so this attribute must be specified in an `account create` command. This attribute is *not* returned by an `account show` command.

(p) `pwdvalid`

A flag set to determine whether the current password is valid. If this flag is set to `no`, the next time a principal logs in to the account, the system prompts the principal to change the password. (Note that this flag is separate from the `pwdexpdate` policy, which sets time limits on password validity.) Possible values are either `yes` or `no`. The default is `yes`.

(q) `renewabletkt`

A flag set to determine if the ticket-granting ticket issued to the account's principal can be renewed. If this flag is set to `yes`, the Authentication service renews the ticket-granting ticket if its lifetime is valid. Possible values are either `yes` or `no`. The default is `yes`.

(r) `server`

A flag set to indicate whether or not the account is for a principal that can act as a server. Possible values are either `yes` or `no`. If the account is for a server that engages in authenticated communications, set this flag to `yes`. If the flag is set to `no`, then clients requests for tickets to contact this principal as a server will be rejected by the security server. The default is `yes`.

(s) `shell`

The path of the shell that is executed when a principal logs in.

(t) `stdtgtauth`

A flag set to determine whether or not service tickets issued to the account's principal use the standard DCE ticket-granting ticket authentication mechanism. Possible values are either `yes` or `no`. The default is `yes`.

The policies are:

(a) `maxtktlife`

The maximum amount of time in hours that a ticket can be valid. When a client requests a ticket to a server, the lifetime granted to the ticket takes into account the `maxtktlife` set for both the server and the client. In other words, the lifetime, cannot exceed the shorter of the server's or client's `maxtktlife`. If you do not specify a `maxtktlife` for an account, the `maxtktlife` defined as registry authorization policy is used.

(b) `maxtktrenew`

The amount of time in hours before a principal's ticket-granting ticket expires and that principal must log in again to the system to reauthenticate and obtain another ticket-granting ticket. The lifetime of the principal's service tickets can never exceed the lifetime of the principal's ticket-granting ticket. The shorter you make this, the greater the security of the system. However, since principals must log in again to renew their ticket-granting ticket, the time needs to take into consideration user convenience and the level of security required. If you do not specify this for an account, the `maxtktrenew` lifetime defined as registry authorization policy is used.

Supports the following operations:

(a) `account catalog`

Returns a list of the names of all accounts in the registry. It takes no arguments. By default, fully qualified names are returned in the form *`cellname/acctname`*. If the `-simplename` option is given then the cellname is not prepended to the front of the account names. The accounts are returned in lexical order.

(b) `account create`

Creates a new account. The argument is a list of names of accounts to be created. Returns an empty string on success. Options are used to specify the attributes of the newly created account. The `group`, `organization`, and `password` attributes *must* be specified on the `dcecp` command line (either in an attribute list or with attribute options). The `password` attribute may *not* be set when `dcecp` is invoked with the `-c` option. All options are applied to all accounts in the argument. Supports both attribute lists (with a `-attribute` option) and attribute options named after each attribute and policy.

This commands requires the user to enter his or her own password to prevent a malicious user who finds a `dcecp` session logged in to a privileged account running from altering an existing account or creating a new account. The users password can be entered via the `-mypwd` option, or if not entered, the user is prompted for it as follows: `Enter Your Password:`. The `-mypwd` option may not be used if `dcecp` was invoked with the `-c` option.

(c) `account delete`

This operation deletes the account from the registry. The argument is a list of names of accounts to be deleted. If the accounts do not exist an error is generated. Returns an empty string on success.

(d) `account generate`

This operation contacts the password generation server to have a new password randomly generated and returned to the user. DCE 1.1 adds a password generation feature. Principals may have a `pwd_val_type` ERA attached to them. This defines if passwords may or must be generated for the principal they are attached too. If the ERA has the appropriate value, the `account generate` command will return a generated password that can be used in an `account modify` command Here are examples based on the three possible values for the `pwd_val_type` ERA:

(i) `mustgen` — Password must be generated.

```
dcecp> set p [account generate melman]
newpwdfoo
dcecp> account modify melman -password $p -mypwd -dce-
dcecp> account modify melman -password foo -mypwd -dce-
Error: Password must be randomly generated.
dcecp>
```

(ii) `cangen` — Password may be generated.

```
dcecp> set p [account generate melman]
newpwdfoo
dcecp> account modify melman -password $p -mypwd -dce-
dcecp> account modify melman -password foo -mypwd -dce-
dcecp>
```

(iii) `nogen` — Password may not be generated.

```
dcecp> set p [account generate melman]
Error: Password may not be generated.
dcecp> account modify melman -password foo -mypwd -dce-
dcecp>
```

Note that the following should never be done since password will be changed in the account but the user will not know the value of the new password:

```
dcecp> acct mod melman -password [acc gen melman] \-mypwd -dce-
```

(e) `account help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `account modify`

This command is used to change information about an account. The `-add` and `-remove` options are not supported, as each attribute associated with an account must always have a value, however, the `-change` option can be used to modify the value of any one of the attributes. The value of the `-change` option is an attribute list describing the new values for the specified attributes. The `password` attribute may *not* be set when `dcecp` is invoked with the `-c` option.

This commands requires the user to enter his or her own password to prevent a malicious user who finds a `dcecp` session logged in to a privileged account running from altering an existing account or creating a new account. The users password can be entered via the `-mypwd` option, or if not entered, the user is prompted for it as follows: `Enter Your Password:`. The `-mypwd` option may not be used if `dcecp` was invoked with the `-c` option.

(g) `account operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `account show`

This operation returns an attribute list describing the specified accounts. The argument is a list of names of accounts to be operated on. If more than one account is given, then the attributes and policies are concatenated together with a blank line between accounts. Accepts a `-policies` option that returns the polices of the account instead of the attributes. A `-all` option can be used to return the attributes followed by the policies.

Attributes and policies are returned in lexical order. If the account does not have any policies then `nopolicies` is returned.

The policies that are actually in effect can be different from the account policies due to conflicts with registry wide policies. If this is the case, the show command will alter the attribute structure on output to include an `effective` tag and the effective value, much in the same way `organization show` will.

8.11.14. registry

The `registry` object represents the security service database of account information. The registry is a replicated database with a single master copy capable of processing both update and read requests, and zero or more slave copies capable of processing only read requests. The term replica is used to refer to either a master or slave copy of the registry. The commands in `dcecp` automatically bind to a registry server capable of performing a specific operation. For most operations, the user does not care which registry server is bound to, as long as the request is fulfilled. However, administrators modifying or checking the replication configuration need to be able to bind to a specific registry. The `registry` command can do this.

After this command executes, the `_b(sec)` convenience variable is set to the name of the server that was bound to for the command. Some commands must bind to the master registry and `_b(sec)` will be set to reflect this. All of the commands can accept an argument, and some require one. The argument is the name of a single registry server to contact. The format of an argument and of the value of `_s(sec)` variable are the same and can be in one of the following formats:[25]

(a) A specific cell name (or `/.:` for the local cell) to bind to any replica in the named cell.

(b) The global name of a replica to bind to that specific replica in that specific cell.

(c) The name of a replica as it appears on the replica list to bind to that replica in the local cell.

(d) A string binding to a specific replica (e.g., `ncadg_ip_udp:15.22.144.163`), see the *Bindings* section for the two available syntaxes for string bindings. This form is used primarily for debugging or if the Cell Directory Service is not available.

(e) If no argument is given, then the value of `_s(sec)` is used. If the variable is not set, then the default argument of `/.:` is assumed.

Some commands require an argument or only accept some of these formats so that any ambiguities are avoided. Arguments supersede the setting of `_s(sec)`. The argument is used explicitly, not as a hint. If the specified server cannot handle the request, an error will be returned. The argument may *not* be a list of registries.

---

25. The security API is forgiving. You can enter a cellname followed by an invalid name and you will bind to a registry server in the cell. The name that the commands set the `_b(sec)` variable to is always in the form of a server entry name. For example, if you set `_s(sec)` to `/.../`*`validcellname`*`/bad/object/name` in a cell with only one registry server, then `_b(sec)` will be set to `/.../`*`validcellname`*`/subsys/dce/sec/master` by the registry commands.

Other `dcecp` objects such as: `principal`, `group`, `organization`, and `account` are used to represent most of the data in the registry. The `registry` object is only used for security replication configuration (i.e., `sec_admin` functionality) and the manipulation of registry-wide attributes (a.k.a. properties) and policies. Registry-wide policies refer to policies for organizations (standard policies in `rgy_edit` terminology) and accounts (see the `rgy_edit auth policies` command). For a list of the policies supported, see the descriptions of both the `organization` and the `account` objects.

There are two ACL's used to allow `registry` operations to be performed. For those that have to deal with replication the `replist` object's ACL (usually `/.:/sec/replist`) controls access. For those that deal with registry attributes and policies, the `policy` object's ACL (usually `/.:/sec/policy`) controls access. All of the `registry` operations described below deal with replication information, and as such access to them is based on the `replist` object, except for the `modify` and `show` operations when they are dealing with non-replication attributes.

The list of attributes is as follows:

(a) `deftktlife`

The default lifetime (in hours) for tickets issued to principals in this cell's registry. The value is an integer.

(b) `mingid`

The starting point for gid's automatically generated by the Security Service when a group is created. You can explicitly enter a lower gid than this number; it applies only to automatically generated numbers. The value is an integer.

(c) `minorgid`

The starting point for orgid's automatically generated by the Security Service when an organization is created. You can explicitly enter a lower orgid than this number; it applies only to automatically generated numbers. The value is an integer.

(d) `minuid`

The starting point for uid's automatically generated by the Security Service when a principal is created. You can explicitly enter a lower uid than this number; it applies only to automatically generated numbers. The value is an integer.

(e) `maxuid`

The highest number that can be supplied as a uid when principals are created. This maximum applies to both the system-generated and user-entered uid's. The value is an integer.

(f) `mintktlife`

The minimum amount of time (in minutes) before the principal's ticket must be renewed. The value is an integer. This renewal is performed automatically with no intervention on the part of the user. The shorter this time is, the greater the security of the system. However, extremely frequent renewal can degrade system performance. Both system performance and the level of security required by the cell should be taken into consideration when selecting the value of this attribute. This is a registry-wide value only, it cannot be set for individual accounts.

(g) `hidepwd`

Determines whether encrypted passwords are displayed or not. If this attribute is set to `yes`, an asterisk is displayed in place of the encrypted password in command output and files where passwords are displayed. Possible values are either `yes` or `no`.

Several of the following attributes describe sequence numbers. Sequence numbers are displayed as two integers separated by a period, e.g., *`high.low`*. Both are 32 bit integers, the *`high`* is only incremented when the *`low`* wraps.

Each registry replica keeps the following propagation information concerning registry updates:

(a) `name`

The name of the replica. It is in the form of a fully qualified CDS name.

(b) `type`

Indicates if the replica is a `master` or a `slave`.

(c) `cell`

The name of the cell that the replica is in. It is a fully qualified cellname.

(d) `uuid`

The UUID of the replica.

(e) `status`

The state of the replica, one of:

(i) `becomingmaster`

The replica is in the process of becoming a master.

(ii) `becomingslave`

The replica is a master in the process of becoming a slave

(iii) `changingkey`

The replica is in the process of having its master key changed.

(iv) `closed`

The replica is in the process of stopping.

(v) `copyingdb`

The replica is in the process of initializing (copying its database to) another replica.

(vi) `deleted`

the replica is in the process of deleting itself.

(vii) `disabled`

The replica is unavailable for updates but will accept queries. This was called ‘‘in maintenance’’ by `sec_admin`.

(viii) `dupmaster`

Two masters have been found in the cell, and the replica is a duplicate of the real master.

(ix) `enabled`

The replica is available for use. This was called ‘‘in service’’ by `sec_admin`.

(x) `initializing`

The replica is in the process of being initialized by the master replica or another up-to-date replica.

(xi) `savingdb`

The replica is in the process of saving its database to disk.

(xii) `unavailable`

The replica cannot be reached.

(xiii) `uninitialized`

The database is a stub database that has not been initialized by the master replica or another up-to-date replica.

(xiv) `unknown`

The replica is not known to the master.

(f) `lastupdtime`

The localized date and time the replica was last updated.

(g) `lastupdseq`

The sequence number of the last update the replica received.

(h) `addresses`

A list of the network addresses of the replica. Can be more than one for connectionless and connection-oriented protocols for example.

(i) `masteraddrs`

The network address of the master registry replica of the cell. This is what the replica believes, it is not necessarily correct. Can be more than one for connectionless and connection-oriented protocols for example.

(j) `masterseqnum`

The master sequence number, which is the sequence number of the event that made the replica the master. This is what the replica believes, it is not necessarily correct.

(k) `masteruuid`

The UUID of the master registry replica of the cell. This is what the replica believes, it is not necessarily correct. The value is a UUID.

(l) `version`

The version of the security server software.

(m) `updseqqueue`

The update sequence numbers that are still in the propagation queue and have yet to be propagated. This is only present in the master replica. This is a list of two numbers, the first is the base propagation sequence number, i.e., the last number known to have been received by all replicas. The second is the sequence number of the last update made on the master.

The master replica keeps the following for *each* replica:

(a) `name`

The name of the replica. It is in the form of a fully qualified CDS name.

(b) `type`

Indicates if the replica is a `master` or a `slave`.

(c) `propstatus`

The status of the propagation, possible values:

(i) `delete`

The replica is marked for deletion.

(ii) `initmarked`

The replica is marked for initialization, that is getting an up-to-date copy of the registry.

(iii) `initing`

The replica is in the process of initialization, that is getting an up-to-date copy of the registry.

(iv) `unavailable`

The replica cannot be reached.

(v) `update`

The replica is ready to receive propagation updates.

(d) `lastupdtime`

The localized time the last update was sent to the replica.

(e) `lastupdseqsent`

The last sequence number of the last update sent to this replica. The value is an integer.

(f) `numupdtogo`

The number of outstanding updates. The value is an integer.

(g) `lastcommstatus`

The status of the last communication with the replica.

Supports the following operations:

(a) `registry catalog`

Returns a list of the names of the security servers (i.e., each copy of the registry) running in the cell. This is also known as the replica list. The order is arbitrary. The `_s(sec)` variable may be set to point to another cell. The optional argument can specify the name of one other cell or a single string binding.

(b) `registry connect`

Connects the local (default cell of local host) to the cell specified by argument. The argument is the fully qualified name of a single cell, it is not a list or a string binding. It creates an account in local cell for the foreign cell and creates an account in the foreign cell for the local cell, both with the same key.

This command requires the following options to specify information in the local cell: `−group`, `−org`, `−mypwd`. It also requires the following options to specify information for the foreign cell: `−fgroup`, `−forg`, `−facct` and `−facctpwd`. Additionally, the `−expdate` is optional accepted to specify the expiration date of each account. Each account would be set to expire at the same time.

This command creates the group and organization specified as the values of the four relevant options if necessary and puts the relevant principal in them if necessary. Be careful on typing, if the group or organization are misspelled they will be created. This is the same behavior as `user create −force`. The reason is cleanup if one group works and the second (foreign) fails, having to cleanup the local case would be difficult.

(c) `registry delete`

Called with no options this command performs an orderly deletion of a security replica specified as the argument. That is, it binds to the master registry, marks the specified replica as deleted, propagates this deletion to the other replicas, sends a message to the specified replica to delete itself, and finally removes the replica from the masters replica list. The argument is required and must explicitly identify a security server, not just a cellname. The value of `_b(sec)` is set to the master server contacted.

If the specified replica is unreachable, the `−force` option can be used. This option causes the master to delete the specified replica from the replica list and then propagates that deletion to the other replicas on the list. The specified replica is not contacted. When that replica is again accessible, the `registry delete` command should be called, specifying the replica and using the `−only` option. This causes that replica to delete its copy of the registry database and stop running. If `−only` is used `_b(sec)` is set to the name of the replica contacted. The `-force` and `-only` options may not be used together.

Returns an empty string on success.

(d) `registry disable`

Disables the *master* registry for updates. This is called `maintenance mode` in `sec_admin`. The argument is a single name of a registry master to be disabled. Read-only copies are not allowed, but the name of other cells can be used. If no argument is given, the value of the `_s(sec)` variable is used. If that is not set, the master registry of the machines default cell is used. Returns an empty string on success.

(e) `registry dump`

Returns the replica information for each replica in the cell. It takes no arguments and `_b(sec)` is set to the name of the last replica displayed. Replicas are displayed with a blank line between them. This command is the same as the following script:

```
foreach i [registry catalog] {
  lappend r [registry show $i -replica]
  append r
}
return r
```

(f) `registry enable`

Enables the *master* registry for updates. This is called `server mode` in `sec_admin`. The argument is a single name of a registry master to be enabled. Read-only copies are not allowed, but the name of other cells can be used. If no argument is given, the value of the `_s(sec)` variable is used. If that is not set, the master registry of the machines default cell is used. Returns an empty string on success.

(g) `registry help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(h) `registry modify`

This command is used to change attributes and policies of the registry. The master registry is always the one contacted to fulfill the request. If no argument is given, `_s(sec)` is used as a hint, if it is a replica, the master of the cell of the replica is contacted. If an argument is specified it is assumed to be an explicit request to contact that server, if a replica is specified and not a master, an error is returned. Returns an empty string on success.

The `-change` option can be used to modify the value of any one of the above listed attributes and policies. The value of the `-change` option is an attribute list describing the new values for the specified attributes. Also supports attribute options. The `-add` and `-remove` options are not supported as each of the above attributes and policies must always have a value.

Each replica (master and slave) maintains its own master key used to access the data in its copy of the database. This command accepts the `-key` option to generate a new master key for the replica named in the required argument and to re-encrypt account keys using the new key. The new master key is randomly generated. If `-key` is used other options may *not* be used. This option requires that a single registry be named as an argument.

(i) `registry operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(j) `registry set`

Changes which replica is the master. The argument is the name of a single registry replica. If called with no options, the command changes the replica named in the argument to be the new master for the cell and changes the existing master to be a slave. Can also be called with either a `-slave` or `-master` option in which case it changes the type of replica specified as an argument to what the option specifies. If the `-master` option is used, the system checks other slaves to see if they are more up-to-date than the one specified in the argument. If any are, the command fails with an error unless the `-force` option is used.

The value of `_b(sec)` is set in the following ways. If called with no options, then it is set to specify the old master replica. If called with either `-slave` or `-master` then it is set to the replica specified in the argument. Returns the empty string on success.

(k) `registry show`

Returns information about the registry and its replicas. Takes an optional argument specifying a single registry replica to contact. Specifying an argument is the same as setting the `_s(sec)` convenience variable before executing a `registry show` command without any argument.

Returns a variety of different information based on the option given. If called with no options or with the `-attributes` option, returns an attribute list in alphabetical order of all the registry-wide attributes. If called with the `-xattrs` option then Extended Registry Attributes are returned instead of the registry-wide attributes. If called with the `-policies` option, returns an attribute list in alphabetical order of all the registry-wide polices. For example:

```
dcecp> rgy sho -at
{deftktlife +0-10:00:00.000I-----}
{hidepwd yes}
{maxuid 32767}
{mingid 5100}
{minorgid 100}
{mintktlife +0-00:05:00.000I-----}
{minuid 12429}
dcecp> rgy sho -po
{acctlife unlimited}
{maxtktlife +1-00:00:00.000I-----}
{maxtktrenew +28-00:00:00.000I-----}
{pwdalpha yes}
{pwdexpdate none}
{pwdlife unlimited}
{pwdminlen 0}
{pwdspaces yes}
```

If called with the `-replica` option, returns the propagation information that is kept by the replica specified in the argument or in `_s(sec)`. For example:

```
dcecp> rgy sho -re
{name /.../absolut_cell/subsys/dce/sec/master}
{type master}
{cell /.../absolut_cell}
{uuid 91259b6c-9415-11cd-a7b5-080009251352}
{status enabled}
{lastupdtime 1994-07-05-14:38:15.000-04:00I-----}
{lastupdseq 0.191}
{addresses
 {ncacn_ip_tcp 130.105.5.93}
 {ncadg_ip_udp 130.105.5.93}}
{masteraddrs
 {ncacn_ip_tcp 130.105.5.93}
 {ncadg_ip_udp 130.105.5.93}}
{masterseqnum 0.100}
{masteruuid 91259b6c-9415-11cd-a7b5-080009251352}
```

```
{version secd.dce.1.0.2}
{updseqqueue {0.187 0.191}}
```

Can also be called with the `-master` option, in which case the master is contacted and the propagation information that the master keeps for each slave is returned. In this case the argument if present, be a master or a cellname or an error is returned. If there is no argument the master of the local cell is contacted.

All the options to this command are mutually exclusive.

(l) `registry stop`

Stops the security server process specified in the argument. The argument is required and must explicitly identify one replica (i.e., just a cellname is not valid). The `_b(sec)` variable is set to the name of the specified replica.

(m) `registry synchronize`

Causes the specified replica to reinitialize itself with an up-to-date copy of the database. The required argument is a name of a single registry replica to operate on. This command binds to the master, marks the specified replica for reinitialization, sends a message to the replica informing it to reinitialize itself and providing a list of other replicas with up-to-date copies of the registry. The specified replica then selects a replica from the list and asks for a copy of the database. The `_b(sec)` variable is set to the name of the *master* replica contacted.

This command is generally not used under normal circumstances. Mainly because the registries should remain synchronized. Also there will be functionality in the DCE 1.1 `secd` such that if the master loses contact with a slave for a protracted period of time, the master reinitializes the slave automatically by giving it a fresh copy of the entire database when communication is restored. In pre-1.1 versions of DCE, if the master loses contact with the slave you must explicitly use the `registry synchronize` or the `sec_admin initrep` command to perform this function.

(n) `registry verify`

Checks if all registry replicas are up-to-date. Returns and empty string if they are. If not, a list of out-of-date and unreachable replica names (or bindings if names are not available) is returned. This elements of this list are suitable use in a `foreach` loop around a call to a `registry synchronize` command.

8.11.15. xattrschema

The `xattrschema` object represents the schema information for an extended registry attribute (ERA). Extended registry attributes may be attached to principals, groups, organizations, and registry-wide policies, as well as on server configuration and server execution objects supported by `dced`. The operations of other `dcecp` objects can manipulate ERA instances, this object is used to manipulate the definition of the schema of these attributes.[26]

Each ERA that can be added to some object, has a type and possibly a value. The definition of the type is stored in the schema. Each ERA type has one ERA schema object associated with it. For ERA's in the registry, these schema objects have the name: `/.:/sec/xattrschema/`*`typename`*.[27] The argument to these commands is a list of names of schema objects of this form. For ERA's in `dced` these schema objects have the name: `/.:/hosts/`*`hostname`*`/xattrschema`.

Each ERA type is defined to be attachable to a specific object type. The object type is identified by the UUID of the ACL manager for that object type. This ensures that the naming mechanism used to identify the object for purposes of manipulating its ACL is the same as that used to manipulate its ERA's.

---

26. The word ''attribute'' is very overused in this section, but the meaning should be clear by context. An ERA on an object is referred to as an ERA. Each ERA type has a schema entry. Each schema entry has various attributes.

27. The `/.:/sec` is not fixed, it can be different for each cell, though this is the default setting as shipped by OSF.

Each ERA schema object has the following attributes:

(a) `aclmgr`

A set that lists the ACL managers that support the object types on which ERA's of this type can be created. For each ACL manager type, the permissions required for attribute operations are also specified. Each acl manager is described with a list, the format is:

`uuid queryset updateset testset deleteset`

where the first is the UUID of the ACL manager, and the rest are the sets of permissions (contentated permission strings as found in an ACL) required to perform each type of operation. The value of this attribute is actually a list of these lists. For example:

```
{8680f026-2642-11cd-9a43-080009251352 r w t D}
{18dbdad2-23df-11cd-82d4-080009251352 r w t mD}
```

This attribute is modifyable after creation, but only in a limited way. New ACL managers can be added, but existing ones cannot be removed or changed. This attribute is required but has no default value.

(b) `annotation`

This is a comment field used to store information about the schema entry. It is a PCS string, the default is the empty string.

(c) `applydefs`

Indicates that if an ERA on a given principal is not found, the related organization (of the account) should be check for that ERA and it's value used.[28] If set to `no`, an attribute query will return an attribute instance only if it exists on the object named in the query. The possible values are either `yes` or `no`, the default is `no`.

This attribute is only advisory in DCE 1.1. Future versions of DCE will support this functionality.

(d) `encoding`

The data type of the ERA value. This attribute is not modifiable after creation. It is a required attribute and has no default value. Legal values are one of:

(i) `any`

The value of the ERA can take on any encoding. This encoding type is only legal for the definition of an ERA in a schema entry. All instances of an ERA must have a concrete encoding type (i.e. any encoding except `any`).

(ii) `attrset`

The value of the ERA is a list of attribute type UUID's used to retrieve multiple related attributes by specifying a single attribute type on a query.

(iii) `binding`

The value of the ERA contains authentication, authorization and binding information suitable for communicating with a DCE server.

The syntax is a list of two elements. The first is a list of security information where the first element is the authentication type, either `none` or `dce`, followed by information specific for each type. The type `none` has no further info. The type `dce` is followed by a principal name, a protection level (one of `default`, `none`, `connect`, `call`, `pkt`, `pktinteg`, or `pktprivacy`), an authentication service (one of `default`, `none`, or `secret`), and an authorization service (one

28. In future versions of DCE, ERA's will be able to be attached to the policy object, and used as cell-wide defaults for all types of objects.

of `none`, `name`, or `dce`). Three examples of the security information list are:

```
{none}
{dce /.:/melman default default dce}
{dce /.:/melman pktprivacy secret dce}
```

The second is a list of binding information, where binding information can be string bindings (see the *Bindings* section), server entry names or binary representations of the network representation of a tower. Three examples of binding information are:

```
{/.:/hosts/hostname/dce-entity
 /.:/subsys/dce/sec/master}
{ncadg_udp_ip:130.105.96.3[1234]
 ncadg_udp_ip:130.105.96.6[1234]}
{00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f \
10 11 12 13
 22 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f \
12 11 12 13}
```

(iv) `byte`

The value of the ERA is a string of bytes. The byte string is assumed to be pickle or is otherwise a self describing type. It is expected that attributes of this type will usually not be entered manually. The format of output is hex bytes separated by spaces with 20 bytes per line.[29] For example, suppose the attribute name was `bindata`:

```
{bindata
 {00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f \
10 11 12 13
  22 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f \
12 11 12 13}}
```

The braces indicate that `bindata` has one value. Passing the above attribute to `llength` would return `2`. On input all white spaces will be compressed so that users may enter the data as bytes or words or any combination, whichever is more convenient. Therefore a user could enter the following as input:

```
{bindata
 {00010203 0405 06070809 0a0b 0c0d0e0f 10111213
  22212223 2425 26272829 2a2b 2c2d2e2f 12111213}}
```

(v) `i18ndata`

The value of the ERA is a string of bytes with a tag identifying the (OSF registered) codeset used to encode the data. See [RFC 40]. The syntax is a Tcl list where the first element is the codeset identifier; it is either a 32-bit integer or string name as obtained from the codeset registry. The second element of the list is the actual string in the specified codeset.

(vi) `integer`

The value of the ERA is a signed 32 bit integer.

(vii) `printstring`

---

29. Tcl allows binary data to be entered via backslash substitution using one of two notations: `\x`*hh* or `\`*ooo*. These are not used to display binary data by `dcecp` so that Tcl will not transform the data while manipulating it. For example, if an ERA was returned as `\x41`, a variable set to that returned value, such as `_r` would pass it through the interpretor and have backslash substitution performed and be set to the string `A`.

The value of the ERA is a printable IDL character string using PCS.

(viii) `stringarray`

An array of PCS strings; represented as a Tcl list of strings.

(ix) `uuid`

The value of the ERA is a UUID.

(x) `void`

The ERA has no value. It is simply a marker that is either present or absent. ERA's of this encoding type are displayed as having a value `present`. This is so they are easier to deal with in an attribute list format. If the ERA is not present on an object then it is not displayed at all, as opposed to being displayed with a value `notpresent`.

The value `present` should be used when adding the ERA via either the `create -attribute` or the `modify -add` constructs. It may be used for the `modify -remove` construct but is not necessary. If the value is not used it means to remove the entire attribute, if it is used it means remove the value from the attribute, but since it is in some sense the last value of the attribute, the attribute is removed. The `modify -change construct is unimportant since the ERA has no value and cannot be changed.`

`Pseudo examples follow:`

```
dcecp> pgo create -attribute {void_era present}
dcecp> pgo modify -add {void_era present}
dcecp> pgo modify -remove {void_era}
dcecp> pgo modify -remove {void_era present}
dcecp> pgo show  # if ERA is not present
dcecp> pgo show  # if ERA is present
{void_era present}
```

(e) `intercell`

Specifies the action that should be taken by the Privilege Server when reading ERA's from an EPAC from foreign cell. Possible values are:

(i) `accept` — accepts ERA's from foreign cells. The only check applied is uniqueness if indicated by the `unique` attribute.

(ii) `reject` — discards ERA's from foreign cells. This is the default.

(iii) `evaluate` — invokes a trigger function to a server that will decide whether the ERA should be kept, discarded or mapped to another value.

This attribute is only advisory in DCE 1.1. Future versions of DCE will support this functionality.

(f) `multivalued`

Indicates that ERA's of this type may be multi-valued (i.e., multiple instances of the same attribute type may be attached to a single object). The possible values are either `yes` or `no`. This attribute is not modifiable after creation and defaults to `no`.

(g) `reserved`

If set then this schema entry may not be deleted through any interface by any user. The possible values are either `yes` or `no`. It defaults to `no`.

(h) `scope`

The value of this attribute is the name of a security directory or object in the registry. If it is an object, then only that object may have instances of this ERA attached to them. If it is a directory, then only descendants of this directory may have instances of this ERA attached to them. The default is an empty string which does not limit which objects ERA's may be attached to. E.g., if this attribute is set to `principal/osf/dce` only principals with a prefix of `osf/dce` in the name may have this type of ERA.

This attribute is only advisory in DCE 1.1. Future versions of DCE will support this functionality.

(i) `trigtype`

Identifies if there is a trigger and if so what type it is. The possible values are one of: `none`, `query`, or `update`. If this attribute is anything other than `none`, then `trigbind` must be set. This attribute is not modifiable after creation and the default is `none`.

(j) `trigbind`

Contains binding information for the server that will support the trigger operations. This field must be set if `trigtype` is not `none` or if `intercell` is set to `evaluate`. The value of this attribute is of the format described by the `binding` encoding. It defaults to the empty string.

(k) `unique`

Indicates that each instance of the ERA must have a unique value within the cell for a particular object type (e.g., principal). The possible values are either `yes` or `no`. This attribute is not modifiable after creation and defaults to `no`.

This attribute is only advisory in DCE 1.1. Future versions of DCE will support this functionality.

(l) `uuid`

The internal identifier of the ERA. The value is a UUID. This attribute is not modifiable after creation. If not specified on the create command, a value is generated by the system.

Supports the following operations:

(a) `xattrschema catalog`

This operation returns a list of all the schema entries defined in the service. The service is specified by an argument as on other `xattrschema` operations. Returns fully qualified names by default. The `-simplename` option will return only the last component of the names.

(b) `xattrschema create`

Creates a new schema entry for an ERA. The argument is a list of names of schema entries to be created. Attributes for the created schema entries can be specified via attribute lists (with a `-attribute` option) and attribute options. All atributes are applied to all entries to be created. If a name is given, the `uuid` attribute may not be specified since each ERA must have a unique UUID. Returns an empty string on success.

(c) `xattrschema delete`

This operation deletes a schema entry. The argument is a list of names of schema entries to be deleted. If they do not exist an error is generated. This command also deletes all ERA instances of the schema entry. Returns an empty string on success.

(d) `xattrschema help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `xattrschema modify`

This operation is used to change attributes of schema entries. The argument is a list of names of schema entries to be operated on. All modifications are applied to all schema entries named in the argument. Schema entries are modified in the order they are listed and all modifications to an *individual* schema entry are atomic. Modifications to multiple schema entries are not atomic. A failure for any one schema entry in a list causes an error to be generated and the rest of the operation to be aborted. Returns an empty string on success.

The `-change` option can be used to modify the value of any one of the attributes that is described as modifiable above.[30] Its value is an attribute list describing the new values for the specified attributes.

Attribute options are supported as well.

(f) `xattrschema operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `xattrschema rename`

This operation changes the name of a specified ERA. The argument is a single name of an ERA to be renamed. Takes a required `-to` option with a value of the new name. The value may not be a list. Returns an empty string on success.

(h) `xattrschema show`

This operation returns an attribute list describing the specified schema entries. The argument is a list of names of schema entries to be operated on. If more than one schema entry is given, then the attributes are concatenated together with a blank line between schema entries. Attributes are returned in alphabetical order.

8.11.16. acl

The `acl` object represents an ACL which may be on any object. The `acl_edit` use model is a batched one. Each `acl_edit` session modifies the ACL on one object. The commands are structured so that individual ACL Entries are modified locally, but nothing is written back to the server until either a `commit` or `exit` command is performed. This model will not be followed in `dcecp`. Each `acl` command will write to the server, however, commands will allow more than one modification at once.

ACL's are composed of ACL Entries. ACL Entries are visible only as members of ACL's. There is no object that represents ACL Entries, only the `acl` object representing an entire ACL. Most of the `acl` operations deal directly with the ACL. See the *Data Structures* section for a description of the syntax of ACL's and ACL Entries. An ACL does have one attribute which represents the default cell of the ACL; it is called `cell`. Manipulation of this attribute is possible only through the `modify` and `show` commands.

An object can have more than one ACL. If the object is a container it has three ACL's: the ACL for the container object itself, an ACL specifying the default ACL on new objects added to the container (the initial object ACL), and an ACL specifying the default ACL on new containers added to the container (the initial container ACL). By default the `acl` command will bind to the ACL of the container. The other ACL's can be operated upon by using options on the commands.

In addition there is another ambiguity that can occur in identifying the ACL to operate on. Since entries in the namespace often identify resources that have ACL's (e.g. servers sometimes have ACL's and they export their binding information into an RPC Entry) the name of a object can mean either the object itself or the namespace entry that identifies the object. This ambiguity is also solved by using a option. The options that specify the ACL to edit can be given on any of the operations and are as follows:

(a) `-ic`

Specifies that the command is to operate on the initial container ACL of the named object.

(b) `-io`

30. The `-add` and `-remove` options are not supported, as each attribute associated with a schema entry must always have a value.

Specifies that the command is to operate on the initial object ACL of the named object.

(c) `-entry`

Specifies that the command is to operate on the ACL of the namespace entry of the object.

(d) `-type`

Specifies a particular ACL manager to use. Unneeded in most cases, required where polymorphic objects are supported (in DCE 1.1 this is in principal names and principal directories). Accepts as a value the name of a single ACL manager. This name can be obtained from doing an `acl show -managers` command.

The above options are not mutually exclusive (with the exception of `-ic` and `-io`). For example, to edit the initial object ACL of a principal directory that is also a principal you need to use the `-io` and `-type` options.

The `acl` command is called with one argument which is a list of items specifying the ACL's to be modified. Each item can be either the name (most common) or a list of a string binding (see the *Bindings* section) with a residual name appended to identify the object.[31] Any options used to specify an ACL to operate upon apply to all items entered in the list.

Supports the following operations:

(a) `acl check`

Returns the permissions granted in the specified object's ACL to the principal that invoked the command. The argument is a list of names of ACL's to be operated on. This command accepts the `-entry` and `-type` options, but not the `-io` or `-ic` options. The `check` command returns the permissions the user has to the object, the only ACL needed to resolve this issue, is the permission ACL on the object. The `-entry` option is supported since it describes which object is involved, not merely which ACL on an object is involved.

```
dcecp> acl check /.:/hosts
rwdtcia
dcecp> acl check /.:/subsys /.:/hosts
r--t-ia
rwdtcia
```

(b) `acl delete`

Removes all ACL Entries from the object, except the `user_obj` entry, if it exists. Note that not all acl managers support the `user_obj` ACL entry. If this commands is performed on an ACL which is not capable of having a `user_obj` ACL entry, or just does not have one (resulting in an empty ACL), the correct response from the ACL manager is a bad syntax error.[32] The argument is a list of names of ACL's to be operated on. Generates an error if the ACL does not exist. Returns an empty string on success.

(c) `acl help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `acl modify`

This operation is used to change one or more individual ACL entries. The argument is a list of names of ACL's to be modified. They are processed in the order they are entered. The specific operation to perform is described with one or more of the following options:

---

31. This is used for cell initialization and not too many other times.

32. In general, this operation is not very useful, except for ACL's on file system objects such as files in Episode filesystems.

(i) `-add`

Adds the ACL Entries to the ACL. The value of this option is a list of ACL Entries with permissions filled in.

(ii) `-remove`

Removes the ACL Entries from the ACL. The value of this option is a list of ACL Entries without permissions filled in.

(iii) `-change`

Changes existing ACL Entries in the ACL. The value of this option is a list of ACL Entries with permissions filled in. The permissions will be the new permissions placed on the specified ACL Entries. The ACL Entries must exist in the ACL or an error occurs.

(iv) `-purge`

Purges all masked permissions (before any other modifications are made), in all ACL Entries except: `user_obj`, `other_obj`, `mask_obj`, and `unauthenticated` if they exist. This option is useful only for ACL's that contain an entry of type `mask_obj`.

(v) `-mask`

If a `modify` operation causes a mask recalculation that unintentionally adds permissions to an existing acl entry, the modify operation will abort with an error unless you specify the `-mask` option with a value of either `calc` or `nocalc`, or a unique abbreviation of one of these values.

Specifying `calc` creates or modifies the object's `mask_obj` type entry with permissions equal to the union of all entries other than type `user_obj`, `other_obj`, `mask_obj` and `unauthenticated`. This creation or modification is done after all other modifications to the ACL are performed. The new mask is set even if it grants permissions previously masked out. It is recommended that you use this option only if not specifying it results in an error. If you specify the `calc` option for an ACL that does not support `mask_obj` entry type, an error is returned.

Specifying `nocalc` means that a new mask should not be calculated.

This option is only legal if the `-add` or `-change` option is present on the modify command. The ACL must support the `mask_obj` ACL type, or else an error is returned. If the user explicitly includes a `mask_obj` ACL entry in the command (i.e., in the value of the `-add` or `-change` options), the use of the `-mask` option is illegal. The reason is that the `-mask` option is meant to resolve unintentional results, the presence of a `mask_obj` ACL entry implies intent.

Either all changes are made to an *individual* object, or none are. Multiple actions can be specified on the command line, they will be processed in a fixed order to guarantee proper processing of the ACL's. See [POSIX.6] for a description of this processing order. The command returns an empty string on success. Examples are:

```
dcecp> acl modify /.:/hosts -add {user melman rwcia}
dcecp> acl modify /.:/hosts -change {user melman rwdtcia}
dcecp> acl modify /.:/hosts -add {group dce rwdtcia} \
-remove {user melman rwdtcia}
```

The `modify` operation can change the value of the `cell` attribute using an attribute option called `-cell` and giving the new default cell as the value (it must be one value, not a list):

```
dcecp> acl modify /.:/hosts/ -cell /.../newcell
dcecp> acl modify /.:/hosts/ -add {user melman rwcia} \
-cell /.../newcell
```

The `-cell` option may be combined with other options on a single `modify` command. It is always applied before the other options. Note that changing the default cell of an ACL that has `user` or `group` ACL entries, or their delegate counterparts, can be dangerous. The principal and groups mentioned in these

ACL entries must be in the default cell. If the default cell changes these ACL entries must as well. The `acl modify` command applies the change of cell first and then the change to the ACL entries. This will *not* work if there are ACL entries of the above types, instead the `acl replace` command should be used to change both the cell and the ACL entries at the same time. The `-cell` option of the `acl modify` command can be useful if there are no ACL entries of the above type, or if the ACL is in an incorrect state and must be corrected.

(e) `acl operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `acl permissions`

Returns a list describing the permissions associated with an object. The argument is a list of names of ACL's to be operated on. If more than one is entered the output is concatenated together with a blank line between ACL's. Each element of the list is a list of two items, the first is the permission token, the second is a description of the permission. For example:

```
dcecp> acl permissions /.:/hosts
{r read}
{w write}
{d delete}
{t test}
{c control}
{i insert}
{a administer}
```

(g) `acl replace`

Replaces the entire ACL on the object specified by the argument, with the value of the required `-acl` option. The argument is a list of names of ACL's to be operated on. The syntax of the value of the `-acl` option is a list of ACL entries. Also takes a `-cell` option to specify the new default cell of the ACL. It's value is the name of one cell only (it is not a list). Returns an empty string on success.

(h) `acl show`

Returns a list of the ACL Entries for the specified object. The argument is a list of names of ACL's to be operated on. If more than one is given the output is concatenated together with a blank line between ACL's. If they exist, the `mask_obj` and `unauthenticated` ACL Entries are displayed first. For example:

```
dcecp> acl show /.:/hosts
{unauthenticated r--t---}
{user cell_admin rwdtcia}
{user hosts/absolut/cds-server rwdtcia}
{user hosts/absolut/self rwdtcia}
{user root rwdtcia}
{group subsys/dce/cds-admin rwdtcia}
{group subsys/dce/cds-server rwdtcia}
{any_other r--t---}
```

Note that since UUID's and not names are stored in ACL's, `dcecp` may not be able to determine the name associated with an ACL entry, in this case, the UUID will be returned as the key. For example, this can be caused if the default cell stored in the ACL (also as a UUID) is incorrect, (i.e., it doesn't refer to a real cell), or the users and groups mentioned in `user` and `group` ACL entries aren't registered in the default cell.

If called with the `-cell` option returns the value of the default cell for the ACL, preferrably as a name but as a UUID if this is not possible:

```
dcecp> acl show /.:/hosts -cell
/.../absolut_cell
```

If called with the `-managers` option returns the names of ACL managers supported for the object. These names may be used as the value of a `-type` option to other `acl` commands. The following example uses potentially fictitious results, the names of the ACL managers have not yet been determined.

```
dcecp> acl show /.:/sec/principal/melman -managers
principal secdirectory
```

There is no `-all` option.

8.11.17. dts

The `dts` object represents the `dtsd` process running on a host. Since DTS is designed not to have stored data, this object represents the information in and about a process rather than some data stored somewhere as other objects do. Also this is not implemented as an instance of the `server` object (described below) since objects, not instances, have operations associated with them.

These commands all affect the local `dtsd` entity by default. If given an argument they can affect a remote DCE 1.1 `dtsd`. The argument is a single server entry or string binding representing a `dtsd` that will be contacted for the operation. If the `_s(dts)` convenience variable is set, it is treated as the name of a `dtsd` to contact for this operation. If either of these methods is used, the specified server is the only server that will be contacted in an attempt to complete the operation. The argument on the command line, takes precedence over the value of the `_s(dts)` convenience variable. These commands do *not* set the value of this variable after completion.

There are a number of attributes associated with the `dts` object. All of these can be viewed with the `show` operation, and many can be changed with the `modify` operation. Attribute arguments can contain a maximum of 80 characters and are recalculated to a normalized date format. For example if the input value is `0-0025:10:99.99999999`, the result is `1-01:11:39.990`.

The attributes are:

(a) `autotdfchange`

Specifies whether automatic changes to the time differential factor are enabled or disabled. The value is either `yes` or `no`. The value is determined by the operating system (i.e., it cannot be changed with the `modify` operation).

(b) `clockadjrate`

Specifies the rate at which the DTS server or clerk entity adjusts the node's clock during a synchronization. This may not be set by a user, but is built into `dtsd`.

(c) `clockresolution`

Specifies the amount of time between system clock ticks. The value is determined by the operating system (i.e., it cannot be changed with the `modify` operation).

(d) `globalservers`

Specifies the set of global servers known by the node. The information returned for each server is as follows: the DCE name of the host followed by `/self`, the last time polled, the last observed time, the last observed skew, a binary value of whether the server was used in the last last synchronization, and the transport time. The sub-attributes are respectively called: `name`, `timelastpolled`, `lastobstime`, `lastobsskew`, `inlastsync`, and `transport`.

(e) `globaltimeout`

Specifies the amount of time the node waits for a response to a WAN synchronization request before sending another request or declaring a global server to be unavailable. The number of attempts made to reach the server is controlled by the `queryattempts` attribute. The default value is `0-00:00:15.000`, and the range of possible values is `0-00:00:00.000 - 0-00:10:00.000`.

(f) `localservers`

Specifies the set of local servers known by the node. The information returned for each server is as follows: the DCE name of the host followed by `/self`, the last time polled, the last observed time, the last observed skew, a binary value of whether the server was used in the last last synchronization, and the transport time. The sub-attributes are respectively called: `name`, `timelastpolled`, `lastobstime`, `lastobsskew`, `inlastsync`, and `transport`.

(g) `localtimeout`

Specifies the amount of time the node waits for a response to a synchronization request before sending another request or declaring a server to be unavailable. The number of attempts made to reach the server is controlled by the `queryattempts` attribute. The default is `0-00:00:05.000`, and the range of possible values is `0-00:00:00.000 - 0-00:01:00.000`.

Note that this attribute controls only the initial contact with a time provider. During this initial contact, the time provider itself determines the timeout value for actually reporting back times. This allows a time provider attached to a slow source like a modem to request that `dtsd` wait for a longer interval.

(h) `maxdriftrate`

Specifies the worst-case drift rate of the node's clock, in nanoseconds per second, as determined by the manufacturer's specifications (i.e., it cannot be changed with the `modify` operation).

(i) `maxinaccuracy`

Specifies the inaccuracy limit for the node. When the node exceeds the maximum inaccuracy setting, it attempts to synchronize. The default is `0-00:00:00.100`, and the range of possible values is `0-00:00:00.0 - 10675199-02:48:05.478`.[33]

(j) `minservers`

Specifies the minimum number of servers required for a synchronization. Settings of `1` or `2` may cause unreliable computed times. The default is `3` and the range of possible values is `1-10`.

(k) `nexttdfchange`

Specifies the future time at which the time differential factor is automatically changed. The value is determined by the operating system (i.e., it cannot be changed with the `modify` operation).

(l) `queryattempts`

Specifies the number of attempts that a node makes to contact a server before the node considers the server unavailable. The default is `3`, and the range of possible values is `1-10`.

(m) `syncinterval`

Specifies the interval a node must wait to synchronize. Also specifies synchronization frequency when a node reaches the value specified by the `maxinaccuracy` attribute. For clerks the default is `0-00:10:00.0`, and the range of possible values is `0-00:00:30.0 - 01 00:00:00.00`. For servers the default is `0-00:02.00.0`, and the range of possible values `0-00:00:30.0 - 01 00:00:00.00`.

(n) `tdf`

Specifies the Time Differential Factor (TDF), which is the amount of time the server varies from Greenwich mean time (GMT) or Coordinated Universal Time (UTC). The default is `0-00:00:00.000`, and the range of possible values is `-13-00:00:00 - 13-00:00:00`. This may not be set by a user, but rather

33. This has been changed to be a max of 24 hours as per DCE OT 7357. It really makes no sense for this value to be anything other than seconds.

is obtained from various timezone information repositories (e.g., the `TZ` environment variable, kernel structures, etc.).

(o) `timerep`

Specifies the internal timestamp format used by the node. This is *not* related to the format that is used to display the current time to the user (see the `clock show` command below). Currently DTS only uses `V1.0` timestamps. This attribute may not be set by the user, its value is built into a `dtsd`.

(p) `tolerance`

Specifies the maximum separation allowed between the local clock and the computed time before synchronizations become abrupt rather than gradual (monotonic). The default is `0-00:05:00.000`, and the range of possible values is `0-00:00:00.500 - 10675199-02:48:05.478`.

(q) `type`

Specifies whether the node is a DTS `server` or `clerk`.

(r) `version`

Specifies the DTS software version installed on the node. This attribute cannot be changed with the `modify` operation.

Additional attributes on a server are:

(a) `actcourierrole`

Specifies a server's acting interaction with the set of global servers. The values are the same as for the `courierrole` attribute below. The difference between this attribute and that, is when `courierrole` is `backup`, the courier could be actually acting as a courier or not.

(b) `checkinterval`

Specifies the amount of time between checks for faulty servers. Applicable only to servers that have external time providers. The default is `0-01:30:00.00`, and the range of the possible values is `0-00:00:30.000 - 10675199-02:48:05.478`.

(c) `courierrole`

Specifies a server's interaction with the set of global servers. Possible values are:

(i) `backup`

The local server becomes a courier if none are available on the LAN. This is the default.

(ii) `courier`

The local server synchronizes with the global set of servers.

(iii) `noncourier`

The local server does not synchronize with the global set of servers.

(d) `epoch`

Specifies the server's epoch number. The default is `0`, and the range of possible values is `0-255`. This value may not be changed via the `modify` command; use the `clock set` command with the `-abruptly` and `-epoch` options to change its value.

(e) `lastsync`

Specifies the computed time at the start of the last attempted synchronization.

(f) `provider`

Specifies whether or not the entity used an external time-provider at the last successful synchronization. This attribute applies to servers only and may not be set by a user. The value is either `yes` or `no`.

(g) `serverentry`

Specifies a server's ACL entry name. The default setting is the following recommended value: `/.:/hosts/`*`hostname`*`/dts-entity`.

(h) `servergroup`

Specifies the security group name for the time servers within the cell. The default is `/.:/subsys/dce/dts-servers`.

(i) `serverprincipal`

Specifies a server's principal name for authentication purposes. The default setting is the following recommended value: `/.:/hosts/`*`hostname`*`/self`.

(j) `status`

Specifies the state of the DTS entity. This is a read-only attribute and its possible values are:

(i) `disabled` — The DTS entity is disabled.

(ii) `enabled` — The DTS entity is enabled.

(iii) `syncing` — The DTS entity is synchronizing.

(iv) `updating` — The DTS entity is updating the time.

(k) `uuid`

Specifies the entity's unique identifier, which is generated when the entity is created.

The counters are:

(a) `abrupts`

Specifies the number of times the node clock has been set non-monotonically (abruptly).

(b) `badlocalservers`

Specifies the number of times that a local server was contacted, but it was not in the dts security group.

(c) `badprotocols`

Specifies the number of times the local node failed to process a received message containing an incompatible protocol version.

(d) `badtimereps`

Specifies the number of times the local node failed to process a received message containing an incompatible timestamp format.

(e) `creationtime`

Specifies the time at which the DTS entity was created and the counters were initialized.

(f) `disables`

Specifies the number of times the DTS has been disabled.

(g) `enables`

Specifies the number of times the DTS has been enabled.

(h) `nointersections`

Specifies the number of times the node's time interval failed to intersect with the computed interval of the servers.

(i) `nomemories`

Specifies the number of times the node has been unable to allocate virtual memory.

(j) `providertimeouts`

Specifies the number of times a `dtsd` server process initiated contact with a time provider and did not receive the initial response within the interval specified by the `localtimeout` attribute.

(k) `syncs`

Specifies the number of times the node successfully synchronized.

(l) `syserrors`

Specifies the number of times a DTS process detected a system error.

(m) `toofewservers`

Specifies the number of times a node failed to synchronize because it could not contact the required minimum number of servers.

Additional counters on a server are:

(a) `badservers`

Specifies the number of times that a non-local server was contacted, but it was not in the dts security group.

(b) `diffepochs`

Specifies the number of times the node received time response messages from servers or clerks that had epoch numbers different from its own.

(c) `epochchanges`

Specifies the number of times the server's epoch has changed.

(d) `faultyservers`

Specifies the number of times a server has detected faulty servers (other than itself).

(e) `noglobals`

Specifies the number of times the courier server could not contact any global servers.

(f) `noresponses`

Specifies the number of times the courier server could not contact a specific global server.

(g) `providerfailures`

Specifies the number of times the external time-provider signaled a failure or the node was unable to access the time-provider.

(h) `syncattempts`

Specifies the number of times a server has attempted to synchronize its clock.

Supports the following operations:

(a) `dts activate`

Changes a DTS entity from an inactive state to an active state. The `status` attribute is changed to `enabled`. This tells the DTS entity to begin synchronizing. Takes a `-abruptly` option to determine if the first clock adjustment due to synchronization is an abrupt or gradual one. Returns an empty string on success.

(b) `dts catalog`

Returns a list of the names of all DTS servers in the specified cell. The optional argument is the name of a cell to look in, it defaults to `/.:`. The names are returned in lexical order. The names returned are fully qualified (i.e., they include the cell name) unless the `-simplename` option is used.

(c) `dts configure`

Requires one of two options `-global` or `-notglobal` to configure the local `dtsd` as a global server or not. The difference is whether they are listed in `/.:/cell-profile`. Returns the string `global` or `notglobal` to indicate the current (new) state of the `dtsd`.

(d) `dts deactivate`

Changes a DTS entity from an active state to a deactive state. The `status` attribute is changed to `disabled`. This tells the DTS entity to stop synchronizing. Returns an empty string on success.

(e) `dts help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `dts modify`

Allows attributes to be changed with the `-change` option. Attribute options are also supported for all modifiable attributes. Returns an empty string on success.

(g) `dts operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `dts show`

When called with the `-attributes` returns an attribute list giving the values of the attributes listed above. If called with the `-counters` option then counter information is returned. If called with the `-all` or with both the `-attributes` and `-counters` options then both attribute and counter information is returned. The default behavior (invoked by using no options) is the same as if the `-attributes` option was used. Attributes and counters are listed in the order they are returned by the server.

(i) `dts stop`

This stops the `dtsd` process. Returns an empty string on success.

(j) `dts synchronize`

Causes the local `dtsd` to synchronize with DTS servers. The machine's clock is adjusted accordingly. By default the clock is adjusted gradually. This command also takes the optional `-abruptly` option to have the clock set abruptly. Returns an empty string on success.

8.11.18. clock

The `clock` object represents the clock on a system, and the time that it tells. This object has commands to display and set the time. The time setting commands are done via DTS. The optional argument to these commands is name of a `dtsd` running on some machine. Without an argument the `_s(dts)` convenience variable is checked. If not set, operates on the clock on the local machine.

Supports the following operations:

(a) `clock help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(b) `clock operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(c) `clock set`

Sets the clock to the time specified as the value of the `-to` option. Can also change the epoch of the DTS server via the `-epoch` option. The argument can be the name of a DTS entity to set the clock on another host. If no argument is given the value of `_s(dts)` is checked. If that is not set, the local clock is set. The clock is adjusted gradually by DTS to the specified time. This command takes a `-abruptly` option which specifies that the time is to be immediately changed to the specified time. When this option is present the `-epoch` option must also be specified to indicate a new epoch. Returns an empty string on success. Requires `w` permission on the DTS entity.

(d) `clock show`

Returns a DTS style timestamp with the TDF indicated. Takes arguments as the other operations. Requires `r` permission on the DTS entity.

(e) `clock synchronize`

This is a synonym for the `dts synchronize` command. Takes arguments as the other operations. Returns an empty string on success. Requires `w` permission on the DTS entity.

(f) `clock compare`

Used to compare the difference between the time on a specified machine and a time provider. The optional argument specifies the name of the host to compare time with. If not specified, `_s(dts)` is used. If that is not set, the local clock is checked. A random time provider is used to compare against the specified clock, any DTS entity may be specified to use via the optional `-server` option. Requires `r` permission on the DTS entity.

The output format is as follows:

```
dcecp> clock compare /.:/hosts/ice/dts-entity
{server /.../dcecp_cell.osf.org/hosts/ninja/dts-entity}
{provider yes}
{skew -0-00:00:00.328I-----}
```

8.11.19. aud

The `aud` object represents the audit daemon (called `auditd` in the reference implementation) on a host. The daemon creates audit trails on a single host. The administrative functions are limited to changing the state of the daemon (enabled, disabled, stopped), and changing the strategy used to deal with running out of storage for the audit trails.

If called with no argument, this command operates on the audit daemon on the local host, unless the `_s(aud)` convenience variable is set. If set, the value is taken to be the name of an audit daemon's server entry, as found in the DCE namespace, or as a string binding; and that audit daemon is contacted to service the requests. An argument can also be given to these commands whose value is the same as that of the `_s(aud)` convenience variable. The argument takes precedence over the convenience variable.

The attributes are:

(a) `stostrategy`

The audit trail storage strategy of the daemon. This attribute defines what the daemon will do if the audit trail storage is full. Its possible values are: `save` in which case the daemon will no longer service logging requests (the `state` attribute will be changed to `disabled`) or `wrap` where the daemon will overwrite the old audit trails.

(b) `state`

Tells whether the audit daemon is accepting audit log requests or not. The values are `enabled` or `disabled`.

Supports the following operations:

(a) `aud disable`

Disables an audit daemon. The `state` attribute is changed to `disabled`. The control (`c`) permission on the audit daemon's ACL is required for this command. Returns an empty string on success.

(b) `aud enable`

Enables an audit daemon. The `state` attribute is changed to `enabled`. The control (`c`) permission on the audit daemon's ACL is required for this command. Returns an empty string on success.

(c) `aud help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `aud modify`

Allows modification of the attributes above. Accepts the `-change` option which takes an attribute list as a value. Also accepts the attribute options `-stostrategy` and `-state`. The control (`c`) permission on the audit daemon's ACL is required for this command. Returns an empty string on success.

(e) `aud operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `aud rewind`

Rewinds a trail file to the beginning. The argument is the name of an audit daemon to operate on. By default the central trail file is rewound, the `-trail` option can be used to specify another trail file. Returns an empty string on success. The control (`c`) permission on the audit daemon's ACL is required for this command.

(g) `aud show`

Returns an attribute list for the audit daemon. The attributes are returned in lexical order. The read (`r`) permission on the audit daemon's ACL is required for this command.

(h) `aud stop`

Stops the audit daemon process. The control (`c`) permission on the audit daemon's ACL is required for this command. Returns an empty string on success.

8.11.20. audfilter

This object represents audit filters (a.k.a. event filters) which are kept by the audit daemon of each host to determine if an auditable event should be logged. An audit filter consists of a list of guides. The name of the audit filter is a filter type and possibly a key (dependent on the type).

This command operates on the audit daemon on the local host, unless the `_s(aud)` convenience variable is set. If set, the value is taken to be the name of an audit daemon's server entry, as found in the DCE namespace, or as a string binding; and that audit daemon is contacted to service the requests.

The audit filter types are as follows:

(a) `principal` — key is a principal_name
(b) `foreign_principal` — key is a /.../cellname/principal_name
(c) `group` — key is a group_name
(d) `foreign_group` — key is a /.../cellname/group_name
(e) `cell` — key is a cell_name
(f) `cell_overridable` — key is a cell_name
(g) `world` — has no key
(h) `world_overridable` — has no key

So the following are examples of audit filter names: `principal melman`, `group dce`, `world`.

A guide contains three elements: audit conditions, audit actions, and event classes. Each of these three elements is itself a list. Event classes are definable by the administrator. The possible audit conditions are: `success`, `failure`, `denial`, and `all`. The possible audit actions are: `none`, `log`, `alarm`, and `all`. If the `all` bit is returned from the API, then *only* `all` is displayed by the `show` operation. I.e., you will not see `{log alarm all}`.

Supports the following operations:

(a) `audfilter catalog`

Returns a list of names of all filters in the audit daemon. This command takes no arguments. The names are a list of a type and if necessary a key. They are returned in an arbitrary order. The read (`r`) permission on the audit daemon's ACL is required for this command. In the following example, there are four filters in the audit daemon:

```
dcecp> audfilter catalog
{principal melman}
{foreign_principal /.../cell_X/kevins}
{group dce}
world
```

(b) `audfilter create`

Creates a new audit filter. The argument is a list of names of audit filters to be created. Since a filter that has no guides will be removed by the audit daemon during a ''garbage collect'' phase, this command requires a `-attribute` option whose value is a list of guides to be added to the specified audit filters on creation. All guides are added to all audit filters specified to be created. Returns an empty string on success.

(c) `audfilter delete`

Deletes the filter including all filter guides. The argument is a list of names of audit filters to be deleted. The write (`w`) permission on the audit daemon's ACL is required for this command. Returns an empty string on success.

(d) `audfilter help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `audfilter modify`

This operation is used to add or remove one or more guides of a filter. The argument is a list of names of audit filters to be modified. In addition, the specific operation to perform is described with one or more of the following options: `-add` and `-remove`. The values of these is a list of guides. If more than one guide is specified then all guides are operated on, but *not* atomically. If the last guide is removed from a filter, the filter will be deleted at some point by the audit daemon. Returns an empty string on success. The write (`w`) permission on the audit daemon's ACL is required for this command. For example:

```
dcecp> audfilter modify \
> {foreign_principal /.../cell_X/foo} \
> -remove {denial alarm IBM_Confidential_Restricted}
dcecp> audfilter modify world -add {
> denial log Monetary_Transfers}
```

Atomicity of multiple actions is not guaranteed. The effect of removing a guide from a filter which does not have an identical guide is to ''change'' the existing guide(s). Similarly, the effect of adding a guide which partially exists in the specified filter is to ''change'' the existing guide(s). These changes guarantee that the semantics of the removal/addition is maintained.

(f) `audfilter operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `audfilter show`

Returns a list of guides in a specified filter. The argument is a list of filter names (a filter type, and if needed a key) to be shown. If more than one is entered, the output is concatenated together with a blank line between filters. The read (`r`) permission on the audit daemon's ACL is required for this command. For example:

```
dcecp> audfilter show {foreign_principal /.../cell_X/foo}
{denial log IBM_Confidential}
{denial log IBM_Confidential_Restricted}
{denial alarm IBM_Confidential_Restricted}
{denial all IBM_Confidential_Restricted}
dcecp> audfilter show world
{success log Monetary_Transfers}
```

8.11.21. audevents

This object represents the event classes that are recognized by an audit daemon on a host. Each event class is defined in an event class configuration file, and the file name is the symbolic name of the event class.

This command operates on the audit daemon on the local host, unless the `_s(aud)` convenience variable is set. If set, the value is taken to be the name of an audit daemon's server entry, as found in the DCE namespace, or as a string binding; and that audit daemon is contacted to service the requests.

Supports the following operations:

(a) `audevents catalog`

Returns a list of the names of all event classes. This command takes no arguments. The order returned is arbitrary. The read (`r`) permission on the audit daemon's ACL is required for this command. For example:

```
dcecp> audevents catalog
EC_OSF_C2_Authentication
EC_OSF_C2_Object_Deletion
EC_OSF_C2_Security_State
EC_OSF_C2_Controlled_Access
EC_OSF_C2_Network_Exception
EC_OSF_C2_Cryptographic
EC_OSF_C2_Configuration
```

(b) `audevents help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(c) `audevents operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(d) `audevents show`

Returns the contents of an event class. The argument is a list of names of event classes. Returned is a list of one name and several 32 bit integers displayed in hex. The name is the event class name, the rest are the numbers of the events that are in the event class. If more than one event class is given in the argument then several lists of this format will be output, concatenated together as one list with a blank line between event classes.

In the future `create` and `delete` operations could be added to this object to allow for the creation of new event classes. These could be implemented as base Tcl scripts as all that is necessary is the manipulation of files.

8.11.22. audtrail

The `audtrail` objects represents an audit trail file. This object currently supports only one operation which converts the audit trial into a human readable format. Future extensions can add fancier conversion operations, or more likely new options. Supported operations are:

(a) `audtrail help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(b) `audtrail operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(c) `audtrail show`

Returns the audit trail in a human readable format. This command takes as an argument a list of names of audit trial files. If more than one name is given then the output of each audit trail is concatenated together with a blank line between audit trails. Takes one option, `-to` to specify a single filename to put the trail in. If this option is not present, the trail is returned from the command. If the option is present the command returns an empty string. Read permission to the audit files is required for this operation.

8.11.23. log

The `log` object represents the current state of routing for DCE serviceability messages of a process. There are two types of routing supported, serviceability routing and debug routing. Debug routing may be removed from a production environment.

The commands take one argument which is either a fully bound string binding or the name of a RPC server entry. See the *Data Structures* section for a description of the syntax of serviceability routing specifications and debug routing specifications.

Supports the following operations:

(a) `log help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(b) `log list`

Returns a list of serviceability components registered by the server. Takes one argument which a list of either a fully bound string binding or the name of a RPC server entry (they can be mixed in the same list). If more than one is entered the output is concatenated together with a blank line between routings.

Also takes an optional `-comp` option which is used to specify a list of component names. In this case, a list of sub-components is returned where each element of the list is a list of three elements: the subcomponent name, its level, and the text of the long description of the subcomponent. The order is arbitrary. If more than one is entered the output is concatenated together with a blank line between components.

Example:

```
dcecp> log list /.:/hosts/hostname/cds-server
{svc cds dts rpc sec}
```

```
dcecp> log list /.:/hosts/hostname/cds-server -comp dts
{general  0  "General server administration"}
{events   0  "Events received and acted upon"}
{arith    0  "Math operations"}
{ctlmsgs  0  "Control messages received"}
{msgs     0  "Messages received"}
{states   0  "Server state transitions"}
{threads  0  "Thread interactions"}
{config   0  "Server/cell configuration"}
{sync     0  "Server sync interactions"}
```

(c) `log modify`

Is used to change the routing specifications of the specified servers. Takes one argument which a list of either a fully bound string binding or the name of a RPC server entry (they can be mixed in the same list). Only the `-change` option is supported. There are a fixed well-known set of routing defaults. These can be changed, but new routings cannot be added or removed.[34] The `-change` option takes a list of either serviceability or debug routing specifications. Returns an empty string on success.

(d) `log operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(e) `log show`

Returns a list describing the current serviceability routing settings for a server. If more than one is entered the output is concatenated together with a blank line between routings for different servers. The order of the returned routing settings is arbitrary. Takes an optional `-debug` option which indicates that debug routing settings are to be returned instead of normal serviceability routing settings.

8.11.24. server

The `server` object refers to servers running on a host. This one object can affect both the running daemons and the configuration information used by `dced` to start that daemon. The distinction is usually obvious by the definition of the operation or by the name given as an argument. When this is not the case, the ambiguity is resolved by a required option.

Almost all of these commands contact the `dced` on the target host to perform their operations. Exceptions to this are noted below.

The names given to `server` commands are all of the following form: `/.../`*`cell`*`/hosts`*`hostname`*`/config/`*`service`*`/`*`name`*, where *`service`* is one of: `srvrconf`, `srvrexec`, or `server`. The first two umabiguously identifies the correct service as either the configuration service or the exectution service. The third is an ambiguous term but this can usually be resolved by context. For example, the `stop` operation only applies to a `srvrexec` object. In cases where it is still ambiguous, a `srvrconf` object is assumed unless the `-executing` option is present.

Only the ERA's can be modified after creation, other attributes cannot. The attributes are:

(a) `arguments`

The command line arguments passed to program on startup. Its value is a list of strings. May not be modified after creation.

---

34. The length of output on a `show` operation can change, but the semantics are add once then change. To avoid confusion in this case, only `-change` is supported.

(b) `directory`

The working directory that the server is started with. May not be modified after creation.

(c) `entryname`

The name of the RPC entry that the server exports binding information to.

(d) `executing`

A list of two elements, the UUID of the server instance and the pid of the server. This attribute is only present if the server is running. This attribute is multi-valued, one value for each instance of the server.

(e) `gid`

The POSIX gid that the server is started with. May not be modified after creation.

(f) `keytabs`

A list of UUID's of related keytab objects where the server stores its keys. May not be modified after creation.

(g) `operations`

A list of operation id's (integers) supported by the registered server. This attribute is accepted on input but not produced on output. It is implemented mostly for forward compatibility with future versions of `dced`.

(h) `principals`

A list of principal names that the server runs as. For example `secd` runs as three different principals. A fully qualified name is always returned on output. On input a relative principal name represents a principal in default cell of the `dced`. May not be modified after creation.

(i) `program`

The name of the server program to be run. Its value is a string. May not be modified after creation.

(j) `prerequesites`

A list of UUID's of other server configuration objects that represents servers that must be running before this one is started. In DCE 1.1 this information is not used to start the other servers, it is just a note to the administrator. Future version of `dced` will probably take action based on this attribute. May not be modified after creation.

(k) `services`

A list where each element is an attribute list of the following attributes:

(i) `annotation`

A human readable PCS string describing the service.[35]

(ii) `bindings`

A list of string bindings identifying the service. This is only specified for well-known endpoints.

(iii) `entryname`

The of an RPC entry where the service binding information is exports.

(iv) `flags`

35. This is not an i18n string for compatibility with DCE 1.0 endpoint map annotation strings.

The value is a list of key words to identify flags for the server. Currently only one is supported:

[a] `disabled` — The mapping has been removed from the endpoint map.

(v) `ifname`

A PCS name of the interface of the service.

(vi) `interface`

The interface identifier (UUID and version) of the service.

(vii) `objects`

A list of object UUID's supported by the service.

(l) `starton`

This attribute identifies when a server should be started. The value is a list of one more more of the following:

(i) `auto` — Start if an rpc that would be serviced by this server is received by `dced`. Ignored for those servers that are repositories.

(ii) `boot` — Start at system startup.

(iii) `explicit` — Start if `dced` receives a command to start the server (e.g., the `server start` command in `dcecp`).

(iv) `failure` — Start if `dced` detects that the server exited with a non-successful error code.

(v) `schedule` — Reserved for future use, this value will be allowed to specify ‘‘start this server at 6am Sunday morning.’’. The time would have to be specified via an ERA.

May not be modified after creation. Specifying a null value to this attribute means the server will not be started. An example of a possible value is:

```
{server {boot failure auto}}
```

(m) `uid`

The POSIX uid that the server is started with. May not be modified after creation.

(n) `uuid`

The internal identifier of the object. It can be specified on creation, or automatically generated, but once created it may not be modified.

Server configuration objects may also have ERA's attached to them. They may be manipulated by the `modify` operation.

Supports the following operations:

(a) `server catalog`

Returns a list of the names of all server configuration objects on a specified host. If called with the optional `-executing` option, returns the name of all servers known by `dced` that are currently running on the specified host. If called with no arguments returns information about the local host. The optional argument is a list of hostnames. If more than one is specified then the information returned is concatenated together with a blank line between servers. The order of information returned is arbitrary. Fully qualified names are returned by default, the `-simplename` option can be used to remove the `/.../`*`cellname`*`/hosts/`*`hostname`*`/config/`*`service`*`/` prepended on each name from the output.

(b) `server create`

Creates a server configuration object. The argument is a list of names of server configuration objects to be created. A `-attribute` option with an argument list as a value is required to define attributes for the server to be created. Also accepts attribute options. Returns an empty string on success.

(c) `server delete`

Deletes a server configuration object. The argument is a list of names of server configuration objects to be deleted. Returns an empty string on success. An error is returned if any of the objects do not exist.

(d) `server disable`

Disables the specified server. Communicates with `dced` and removes the endpoints for all interfaces registered by the server (except the `rpc_mgmt` interface) from the endpoint map. The argument is a list of names of server execution objects. Accepts an optional `-interface` option to specify a list of interfaces to be disabled. Returns an empty string on success.

(e) `server enable`

Enables the specified server. Communicates with `dced` and adds endpoint mapping for all interfaces registered by the server to the endpoint map. The argument is a list of names of server execution objects. Accepts an optional `-interface` option to specify a list of interfaces to be enabled. Returns an empty string on success.

(f) `server help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(g) `server modify`

Used to add or remove attributes and their values from the server object. The argument is a list of names of server objects to be modified. Accepts the `-change` option which must have an attribute list as its value. Attribute options are *not* supported for this command. The name is always for a server configuration object, you many `not` modify a server execution object.

(h) `server operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(i) `server ping`

Pings a server to see if it is receiving requests. This operation communicates directly with the server and not `dced`[36] The argument is a list identifying the servers to ping. Returns a list of values, one for each server specified in the argument, in the same order. The values are `1` if the server is listening for RPC requests, `0` if not. Each argument can be in one of the following formats:

(i) The name of a server entry in the namespace to be imported from.

(ii) A partially bound string binding with an object UUID specified.

(iii) A fully bound string binding.

(j) `server show`

Returns an attribute list of the server entries specified in the argument. The argument is a list of names of server object entries. If the names are ambiguous, server configuration objects are assumed unless the `-executing` option is present. If the argument is a list the output is concatenated together with a blank line between servers.

```
dcecp> server show /.:/hosts/foster/config/try_tserver
{uuid 003b24d2-a196-1df3-915f-0000c0ba4944}
```

36. Excluding the normal endpoint resolution resulting from getting a partial binding from the namespace.

```
{program tserver}
{arguments /.:/hosts/foster/test_server}
{prerequisites {}}
{keytabs {}}
{entryname /.:/hosts/foster/test_server}
{services
 {{ifname  {test server}}
  {annotation {dcecp server test program}}
  {interface {008bebed-c7c1-1ddc-9cb3-0000c0ba4944 1.0}}
  {bindings {ncadg_ip_udp 130.105.5.50}}
  {objects 0073f23a-2e1a-1ddd-b73a-0000c0ba4944}
  {flags {}}
  {entryname /.:/hosts/foster/test_server}}}
{principals /.../foster_cell/tserver}
{starton boot auto explicit failure}
{uid 0}
{gid 0}
{dir /project/dce/build/nb_486/install/at386/dcetest/\
dce1.1/test/tet/functional/admin/dcecp/ts/server}
```

(k) `server start`

Contacts a `dced` to start a server based on a server configuration object. The argument is a list of names of server configuration objects. Returns the UUID of the started server on success. This is the UUID found in the serverexec object for the server.

(l) `server stop`

Stops the specified running server processes. The argument is a list of names of servers. Returns an empty string on success. Takes an optional `-method` option to specify how `dced` should stop the server. The values of this option must be one of the following:

(i) `rpc` — Have `dced` use `rpc_mgmt_server_stop_listening`. This is the default.

(ii) `soft` — Use a soft local mechanism (e.g., `SIGTERM`).

(iii) `hard` — Use a hard local mechanism (e.g., `SIGKILL`).

(iv) `error` — Use a state-preserving mechanism (e.g., `SIGABRT`).

8.11.25. hostdata

The `hostdata` object represents a hostdata entry stored by `dced` on a host that represents some data, usually a file. The data itself is represented by the `data` attribute of the entry. Remote manipulation of this data is accomplished by these commands. The names of these hostdata objects are in the DCE namespace and are served by `dced`'s. They are of the form: `/.:/hosts/`*`hostname`*`/config/hostdata/`*`name`*. The attributes are:

(a) `uuid`

An internal identifier for the hostdata entry. Its value is a UUID. If not specified on creation, one is generated by `dcecp`. May not be modified after creation.

(b) `annotation`

A human readable comment field limited to PCS data. May not be modified after creation.

(c) `storage`

This is a PCS string that identifies the name of the data repository. In the first release of `dced`, it is a filename. It may not be modified after creation.

(d) `data`

An attribute that represents the actual data. Its syntax is a list of strings. In the common case, each string represents one line in the hostdata file. The data can viewed and modified in two different modes, either as a string, or as binary data.[37] By default the string mode is used, but some of the commands below accept a `binary` option to allow this attribute to be displayed or modified in binary form.

Supports the following operations:

(a) `hostdata catalog`

Returns a list of the names of all objects on the specified host in arbitrary order. The argument is a list of hostnames. If more than one is specified the output is concatenated together with a blank line between hostdata objects. If no argument is given the local host is assumed. Fully qualified names are returned by default, the `-simplename` option can be used to remove the `/.../`*`cellname`*`/hosts/`*`hostname`*`/config/hostdata/` prepended on each name from the output.

(b) `hostdata create`

Creates a hostdata configuration object. The argument is a list of names of hostdata entries to be created. Takes a `-attributes` option to specify configuration information for `dced`. Also accepts attibute options. A `-binary` option can be specified which indicates that the value of the `data` attribute is specified in binary form. Returns an empty string on success.

(c) `hostdata delete`

Deletes a hostdata entry and its data. The arugment is a list of names of hostdata entries to be deleted in the order specified. If the `-entry` option is present, only the configuration information that `dced` keeps is deleted, not the actual hostdata. Returns an empty string on success.

(d) `hostdata help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `hostdata modify`

This operation is used to change attributes of a hostdata entry, including the hostdata itself. The argument is a list of names of hostdata entries to be modified. If more than one is specified all modifications specified are done to each hostdata entry listed. In the first release only the `data` attribute is modifiable, and it can only be completely replaced. Accepts a `-change` option to specify an attribute list of new values. Also accepts attribute options (just `-data` in DCE 1.1). A `-binary` option can be specified which indicates that the value of the `data` attribute is specified in binary form.

(f) `hostdata operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `hostdata show`

Returns an attribute list of the hostdata entries specified in the argument. The argument is a list of names of hostdata entries. If called with the `-entry` option, then the `data` attribute is not returned. A `-binary` option can be specified which indicates that the value of the `data` attribute should be returned in binary form. If the argument is a list the output is concatenated together with a blank line between hostdata objects.

---

37. This is similar to the ASCII and BINARY modes in `ftp`. It's still the same data, it can just be seen in two different ways.

8.11.26. keytab

The keytab object represents key tables (limited to files in DCE 1.1) that store server keys (and key version numbers) on hosts. These keytab's are manipulated remotely via dced and as such there is configuration information that dced maintains about each keytab. This information is represented as the attributes on this object. The keys are considered members of the keytab container.

The names of keytab's are similar to other dced objects, namely: /.../*cell*/hosts/*hostname*/config/keytab/*name*.

A key table has a set of keys. Each key contains a fully qualified principal name, type, version, and value. The value can be created and changed, but is normally not shown on output (see -keys on endpoint show below). Removal of a key is based on the name, type and version number. The Tcl syntax of a key is a list of principal name, type (which is either plain or des), version (a non-negative integer), and value. The value of a des key is 64 bits long and can be represented as ERA's of type byte are. See the xattrschema object for details. The value is valid on input but are not normally displayed so that keys are not shown on the screen. For example:

```
melman des 1 key1
melman plain 3 key2
```

Multiple keys for the same principal are displayed as separate keys. See the example in the keytab show command below.

The following are the attributes of a keytab object:

(a) uuid

The internal identifier of the configuration information kept by dced for this keytab. This is generated automatically by the system if not entered. May not be modified after creation.

(b) annotation

A human readable comment string limited to PCS. May not be modified after creation.

(c) storage

The name of the key table (usually a filename). May not be modified after creation.

(d) data

The contents of the key table. Represented as a list of keys.

Local key tables can also be manipulated without using dced. This is useful in some error recovery situations (e.g., fixing the host keytab when the disk it's on crashes), it is a special dced bypass operation, not a normal keytab operation. All of the following keytab operations take a -local option. When used in this manner dcecp keeps track of the keytab objects that are created on a per session basis. That is, local keytab files are manipulated, but objects representing them that contain all the attributes described above are ''stubbed'' by dcecp. These ''stubbed'' keytab objects have names and attributes just as regular keytab objects do. There is no way for dcecp to know what local key table files exist at startup (they can be anywhere in the file system), so knowledge of them is lost when you exit dcecp.

Key tables store server keys which is sensitive information. It would be a security hole for unauthorized users to obtain this information. As such, the RPC calls used for these operations uses RPC Privacy (i.e., DES encryption) for all the data by default. This is not always possible (e.g., due to encryption export and use laws in various countries). All of the following commands accept a -noprivacy option to use the default protection level instead. This option will probably be deprecated in future releases in favor of a more general mechanism that allows control of all binding handles used by dcecp. See the *Changes for Future Versions* section for a description of the _a convenience variable.

Supports the following operations:

(a) keytab add

This command is used to add a members to a keytab. The argument is a list of names of keytabs to have members added to. The value of the required `-member` option is a list of principal names to be added to each keytab in the argument. The principals must exist or the command will return an error. The principal is added to the key table along with a key. Use either `-random` to have `dced` generate a random `des` key, or `-key` with a value to specify a `plain` key explicitly. The same key (whether specified or randomly generated) is used for all principals being added to all key tables. The `-registry` option will update the principal's key in the registry as well, it is required if `-random` is used. Returns an empty string on success. For example:

```
dcecp> keytab add /.:/hosts/foo/config/keytab/bar \
-member melman -random -registry
dcecp> keytab add /.:/hosts/foo/config/keytab/bar \
-member melman -key keyvalue
```

You must specify either `-registry` or `-version` or both on any `keytab add` command.

(b) `keytab catalog`

Returns a list of the names of all keytab's on the host specified in the argument. If a host name is not specified then the current host is used. If the argument is a list then the output is concatenated together with a blank line between keytabs. The return order is arbitrary. Fully qualified names are returned by default, the `-simplename` option can be used to remove the `/.../`*`cellname`*`/hosts/`*`hostname`*`/config/keytab/` prepended on each name from the output.

(c) `keytab create`

Creates a key table. The argument is a list of names of keytab's to be created. Takes a `-attributes` option to specify configuration information for `dced`. Also accepts attribute options. The contents of the keytab can be specified via the `data` attribute. The value of the option is applied to all elements of the argument list. Returns an empty string on success.

The value of the `data` attribute, if specified, is a list of keys. Each key must have a principal name and key type. The version is optional, if not present, the system will generate a version of `1`. If the key type is `plain` a key value must be specified, if the type is `des` then it may be specified, but if not present, one will be randomly generated only if the `-random` option is present.

For example:

```
dcecp> keytab create /.:/hosts/foo/config/keytab/bar \
-attr {
> {storage /opt/dcelocal/keys/bar}
> {data {melman des}
>       {melman plain 3 key2}
>       {pwang des 2 key3}}}
```

(d) `keytab delete`

Deletes a key table entry and its keys. The argument is a list of names of keytab's to be deleted in the order specified. If the `-entry` option is present, only the configuration information that `dced` keeps is deleted, not the actual key table. Returns an empty string on success.

(e) `keytab help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(f) `keytab list`

Returns a list of all the principals in the specified key table. If the argument is a list of keytab's the output is concatenated together with a blank line between keytabs.

(g) `keytab operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(h) `keytab remove`

This command is used to remove a member from a keytab. The argument is a list of names of keytab's to have members removed from. The value of the required `-member` option is a list of names of principals which will be removed from the keytab's listed in the argument. The two options `-version` and `-type` can be used to limit the keys removed. If either or both of these options is present, then only keys matching the values of these options are removed. The value of the `-version` option can be a list of version numbers. Returns an empty string on success. For example:

```
keytab remove keytabname -member melman -type des
```

removes all `des` keys for principal melman.

(i) `keytab show`

Returns an attribute list of the keytab's specified in the argument. The argument is a list of names of keytab's. If called with `-entry` then the `data` attribute is not returned. If the optional `-members` option is given then only the value of the `data` attribute is returned (a list of keys). Keys are not normally output unless the `-keys` option is used. If the argument is a list the output is concatenated together with a blank line between keytabs. For example:

```
dcecp> keytab show /.:/hosts/foo/config/keytab/bar \
-members
{melman des 1}
{melman plain 3}
{pwang des 2}
```

8.11.27. secval

The `secval` object represents the security validation service running on a host, usually as part of the `dced` server. This service is responsible for maintaining the security credentials of the host machine[38] and is also involved in authenticating local DCE logins and updating the hosts' idea of the cell names. These commands take as an argument a list of names of hosts to operate on.

(a) `secval activate`

Activates a security validation service. If it is not deactivated an error is returned. Returns an empty string on success.

(b) `secval deactivate`

Deactivates a security validation service. If it is not activated an error is returned. Returns an empty string on success.

(c) `secval help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

---

38. There are two reasons to make this a separate object aside from the `server` object with the interface UUID of the security validation service:

(a) The `server disable` command really just removes the endpoint from the endpoint map, the server continues to run, it just won't be routed new requests from new clients (i.e., existing binding handles and clients knowing the endpoint would still work). For `secval` you are really turning off the service.

(b) Since `secval` is implemented in `dced` it registers its endpoints specially in the endpoint map. Normal `server disable` and `server enable` commands will not work for `dced` implemented services.

(d) `secval operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(e) `secval ping`

Validates the credentials returned by a DCE security service. This is expected to be a rarely invoked routine, but could be used to verify that `secd` is trusted. Returns `1` if the credentials are valid and `0` if not. If the argument is a list then a list is returned, with an integer for each server.

(f) `secval status`

Returns a `1` if the secval service is activated, `0` if not. If the argument is a list then a list is returned, with an integer for each server.

8.11.28. uuid

The `uuid` object refers to UUID's. Only two operations are supported:

(a) `uuid create`

Returns a newly generated UUID. Takes no arguments.

(b) `uuid compare`

Compares two UUIDs. Returns `1` if they are equal, `0` if not. This is not just `string compare` since there are some old string formats of UUID's that are supported.

(c) `uuid help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `uuid operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

8.11.29. name

The `name` object refers to DCE names.

(a) `name compare`

Compares two names given as arguments and returns `1` if they both *syntactically* refer to the same name. Otherwise returns `0`.

(b) `name expand`

Takes a single name as an argument and returns the canonical form of the name. This has the affect of converting `/.:` to `/.../`*`cellname`*.

(c) `name get`

Given a single string binding as an argument, returns the fully qualified global name of the host referred to by that binding.

(d) `name help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `name operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(f) `name parse`

Parses a name into a cellname and a residual name. The argument is a single DCE name. Returns a list of two elements: cellname and residual name. A name not begining with a `/` is considered to be a name in the local cell.

8.11.30. utc

The `utc` object represent UTC timestamps and allows manipulation (math and comparison) of them.

(a) `utc add`

Returns the sum of two timestamps. The timestamps can be two relative times or a relative time and an absolute time.

(b) `utc compare`

Takes two timestamps and returns `-1` if the first is earlier, `1` if the second is earlier, and `0` if it is indeterminate. Accepts a `-noinaccuracy` to ignore inaccuracies in comparisons, in this case a return `0` of zero indicates the times are the same.

(c) `utc convert`

Given a timestamp, returns another timestamp expressing the same time in the local timezone. If called with `-gmt` returns GMT formatted timestamp. <

(d) `utc help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(e) `utc multiply`

Accepts two arguments a relative timestamp and an integer or floating point factor and returns former multiplied by the later.

(f) `utc operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `utc subtract`

Returns the difference between two timestamps that express either an absolute time and a relative time, two relative times, or two absolute times.

8.11.31. attrlist

The `attrlist` object represents an attribute as returned or accepted by many `dcecp` command. This object can be use to check or manipulate attribute lists so that they can be used by other commands, most commonly in scripts. The `getvalues` operation will probably be used commonly in interactive sessions, when users want to see the value of one only attribute, since the `show` commands return all attributes.

The arguement to all of these commands is a single attribute list. Supports the following operations:

(a) `attrlist add`

Returns an attribute list with the attributes specified as the value of the required `-member` option appended. For example:

```
dcecp> attrlist add {{a b} {c d}} -member {{e f} {g h}}
{a b} {c d} {e f} {g h}
```

(b) `attrlist getvalues`

Returns the values of all attributes of a type specified by the value of the required `-type` option. The value may only be a single type, but the if the attribute appears more than once in the attribute list, the value of each instance is returned on a separate line though the length of the returned list matches the number of individual values found. For example:

```
dcecp> attrlist getvalues {{a w x} {c y} {a z} -type a
w x
z
```

This command is expected to be use as commonly as `grep` to view the output of `show` commands. For example:

```
dcecp> attrlist get [dir show /.:/hosts] -type CDS_UTS
1994-07-01-10:29:59.265-05:00I0.000/00-00-c0-f7-de-56
```

With abbreviations this could be entered as:

```
dcecp> at g [dir show /.:/hosts] -t CDS_UTS
1994-07-01-10:29:59.265-05:00I0.000/00-00-c0-f7-de-56
```

(c) `attrlist help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `attrlist list`

Returns a list of all the attribute type names in the attribute list in the order that they appear in the list. For example:

```
dcecp> attrlist list {{a b} {c d}}
a b
```

(e) `attrlist modify`

Returns an attribute list with attributes modified as specified by the `-add`, `-remove` and `-change` options. New attributes can be added, or new values added to existing attributes with `-add`. Entire attributes or attribute values can be removed with `-remove`. The `-change` option will remove all values from an existing attribute and replace them with new values specified.

(f) `attrlist operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(g) `attrlist remove`

Returns an attribute list with the attribute types specified as the value of the required `-member` option removed. For example:

```
dcecp> attrlist remove {{a b} {c d} {e f}} -member {c e}
{a b}
```

This command only removes entire attributes, to remove specific values, use the `modify` command.

8.11.32. cdsalias

The `cdsalias` object represents cell names as known by CDS. This object lets you manipulate alias and primary names of DCE cells. Each cell has one primary name. This is the name returned by the cell. Cells may also have alias names. Currently this object only affects the CDS component. The security server and each host must also be informed of any new cell aliases.

This object can also be used to define a hierarchical relation between been one cell and another amongst the CDS' of the two cells. When defining a hierarchical relationship, a trust relationship also needs to be established. Use the `account` object to do this.

Supports the following operations:

(a) `cdsalias catalog`

Returns a list of all defined cell aliases. The first element is the primary name of the cell. Requires read permission on the root directory of the cell. This command takes an optional argument specifying the name of a cell to contact, the argument defaults to `/.:`.

(b) `cdsalias connect`

Establishes a hierarchical relationship between two cells. It takes no argument. The current primary name of the cell is used, the last RDN is removed to identify the parent cell. Requires administer permission on the root directory of the child cell and insert permission on the root directory of the parent directory.

(c) `cdsalias create`

Creates a new alias cellname for the local cell. The required argument is a single fully qualified alias name of the cell. If the alias name already exists an error is returned. Returns an empty string on success. Requires administer permission on the root directory of the cell.

(d) `cdsalias delete`

Deletes an existing alias cellname. The required argument a single fully qualified alias name of the cell. If the alias name does not exist an error is returned. It is an error to attempt to delete the primary cell name. Returns an empty string on success. Requires administer permission on the root directory of the cell.

(e) `cdsalias set`

If called with no options, sets the primary name of the cell to the alias named in the argument. The required argument is a single fully qualified name of an existing cell alias. This command should only be used if the CDS and Security servers as well as each host in the cell know about the cell alias. Requires administer permission on the root directory of the cell. Returns an empty string on success.

8.11.33. Miscellaneous commands

(a) `dcecp_initInterp`

This command will initialize a base Tcl interpreter with all the `dcecp` commands.

(b) `errtext`

Takes a DCE status code as an argument and returns the text of the associated message as found in the message catalogs. The argument can be in decimal, octal (leading `0`), or hexadecimal (leading `0x`) notation.

(c) `login`

Creates a new login context to be used for the rest of the `dcecp` session. Login contexts are stacked. Takes an account name as an argument. The password is prompted for and not echoed to the screen. Accepts a `-password` option to enter a password. Also accepts a `-certify` option to perform a certified login. The `_u` and `_c` convenience variables are updated after a successful login.

(d) `logout`

Logs you out of the current login context as established with a previous `login` command. Only contexts created with `dcecp`'s `login` can be logged out of. Trying to logout of an inherited context results in an error. Leaving `dcecp` will do a `logout` for all contexts created in the session.

(e) `quit`

Exits from `dcecp`. A synonym of the Tcl builtin command `exit`.

(f) `resolve`

Takes a partial string binding and returns a fully bound string binding. Takes a `-interface` option with an interface identifier as an argument to provide enough information for the mapping to occur. Takes an optional `-object` option whose value is a single UUID to specify resolution further.

(g) `shell`

Spawns a command shell for the user. The value of the `SHELL` environment variable is used to obtain the name of the shell to spawn. When the command shell terminates, control is returned to `dcecp`. If called with arguments, they are passed to the shell and executed. Control is returned upon completion. Always returns an emptry string, though an error exception is generated if the shell exists abnormally.

8.12. Tasks

The following are administrator tasks to be implemented as `dcecp` scripts. To the user, they will appear exactly like commands.

8.12.1. user

The `user` object represents all of the data associated with a DCE user. This includes registry information and a CDS directory in the default implementation. The `user` object is meant as a high level abstraction to allow administrators to easily create, delete and view user information.

The `user` object comprises a principal and account in the registry; registers the principal in a group and an organization, creating them if necessary; and a CDS directory named after the principal with the appropriate ACL. Only the principal and account attributes are considered attributes of the `user` object, and are the only ones displayed by the `show` operation.

A principal name is used as the name of user objects. The naming conventions for principal names apply; if no cellname is given it is assumed to be in the current cell. If one is specified, the other cell is contacted.

This object is implemented as a script to allow it to be manipulated and extended on a per-site basis. For example, administrators might want to add GDS and DFS information to the object. Other possible modifications include:

(a) Changing the location of the CDS directory created for users, or remove it completely.

(b) Changing the default ACL's placed on the various objects.

(c) Setting certain attributes or policies on all newly created principals and accounts to match the site's policies.

(d) Setting up site specific defaults for passwords (to be changed by the user later), groups, organizations, principal directories, etc.

(e) Supporting a `user modify` command.

Supports the following operations:

(a) `user create`

Creates a single DCE user. The argument is a list of names of new principals to be added to the registry. Returns an empty string on success. On error attempts to cleanup any interim operations that have completed.

This commands creates a principal and an account for that principal. If either previously exist an error is generated. The principal is then added to the specified group and organization. Since the principal is created, it can't have been a member of either. If the group or organization does not exist an error is generated unless the `-force` option is used. Creates a CDS directory called `/.:/users/`*`principalname`* and add an ACL entry to the default ACL so that the user has `rwtci` permissions on the directory. This allows all access except for deleting the directory and administering replication on the directory.

Attributes and policies for the newly created principal and account may be specified with the `-attributes` option and specifying an attribute list as the value, or with attribute options. This command will attempt to add any unknown attributes as ERA's on the created principal object. Policies of the organization may not be specifed as these would probably affect more than just the created user. The required group and organization names may be specified either as attributes in the `-attributes` option, or via the `-group` and `-organization` options. The required `password` attribute may be provided as in the `account create` command, and the `-mypwd` option is also required.

(b) `user delete`

Deletes a DCE user. Deletes the principal from the registry which also deletes the account and removes the principal from any groups and organizations. Deletes the `/.:/users/`*`principalname`* directory and any contents. The argument is a list of the principal names of DCE users. Returns an empty string on success.

(c) `user help`

Returns help information on the object as described in the *Help System* section. Takes an argument which may be an operation supported by the object or the `-verbose` option to return more information.

(d) `user operations`

Returns a list of the operations supported by the object. It takes no arguments, and always returns a Tcl list suitable for use in a `foreach` statement. The order of the elements is alphabetical with the exception that `help` and `operations` are listed last.

(e) `user show`

Returns the attributes of a DCE user. This includes principal attributes and ERA's, and account attributes and policies. The argument is a list of dce principal names, if more than one is given the output is concatenated together with a blank line between users. The information is returned as if the following commands were run in the following order:

```
principal show
account show -all
```

8.12.2. host

The `host` object represents a machine in (or to be added to) a DCE cell. Currently it represents DCE processes running on a host, but this could be expanded in the future to address static data as well. The `host` object is meant as a high level abstraction to allow administrators to easily configure and start DCE on machines.

The `host` object can configure and start the core DCE services on a machine. This includes `dced`, CDS, and `dtsd`. The argument to this command is the DCE name of a host to operate on. If an argument is not given, then the local host is assumed. If called in this manner then the behavior of the command may be slightly different so that more is done on the local host then is possible remotely. Details are described below in each operation.

This object is implemented as a script to allow it to be manipulated and extended on a per-site basis. For example, administrators might want to add GDS and DFS information to the object. Other possible modifications include:

(a) Adding calls to specialized commands to start or stop application servers. Something like: "printer stop".

Supports the following operations:

(a) `host catalog`

Returns a list of names of hosts in the cell. The names are fully qualified unless the `-simplename` option is used in which case they are cell relative. The optional argument can specify a cell to operate in. For example:

```
dcecp> host catalog
/.../cell_foo/hosts/alpha
/.../cell_foo/hosts/beta
/.../cell_foo/hosts/gamma
dcecp> host catalog -simplename
hosts/alpha
hosts/beta
```

`hosts/gamma`

(b) `host configure`

Configures a single machine named by the argument into a DCE cell. The cell must already exist, having a security and naming service operating. The DCE software must already be installed on the machine. Operates via two mutually exclusive options:

(i) `host configure -client`

Configures the local host as a DCE client machine. The required argument is the fully qualified DCE name of the host (e.g., `/.:/hosts/`*`hostname`*). Also required are the `-secmaster` and `-cds` options to specify the hostname of the security master server and the hostname of any CDS server respectively. Configures the machine to run `dced` (including the secval service), a DTS clerk, `cdsadv`, and `auditd` all via `dced`'s server configuration service. Returns an empty string on success.

(ii) `host configure -server`

Configures the local host as a DCE server machine. It cannot configure a machine as the initial security or cds server in a cell. The required argument is the fully qualified DCE name of the host. If this machine is not already configured into a cell then the equivalent of a `host configure -client` is performed (with the options required by that command required by this command). All configurations are perfromed via `dced`'s server configuration service.

The command requires one or more of the following options: `-security`, `-cds`, or `-dts` to specify which servers to start on the host. The first two take no values. The `-dts` option takes a value of either `local` or `global` to specify what type of server to configure the DTS daemon as. If `-dts` is specified, the command optionally takes a `-courierrole` option which takes the values as per the corresponding attribute on the `dts` object.

(c) `host ping`

This command will test if DCE processes are accesible from the network. It contacts the endpoint mapper (either `rpcd` or `dced`, whatever listens on port 135) on the specified host. The argument is the name or string binding of a single host to ping. Returns `1` if the host responds, `0` if not.

(d) `host start`

Start all DCE processes on the specified host. This command depends on `dced` being running on the specified host, i.e., it may not be used to start DCE on a pre-1.1 machine. The processes that are started are all those listed in the server configuration data stored in the `dced` on the specified host with the `boot` or `explicit` values of the `starton` attribute.

The argument is the name of host to operate on. If called with no argument then `dced` is started on the local host first which requires the appropriate local permissions (usually root). Minimally starts dced, cdsadv, dtsd, and auditd.

(e) `host stop`

Stops running DCE processes on the specified host. This command depends on `dced` being running on the specified host, i.e., it may not be used to stop DCE on a pre-1.1 machine. The argument is the name of host to operate on. The processes that are stopped are as follows in the following order:

(i) All listed in the server execution data stored in the `dced` on the specified host.

(ii) The following commands are then run with an argument if necessary specifying the appropriate host:

```
dts stop
aud stop
registry stop
```

(iii) If called with no argument then `dced` is stopped after all other processes. This might require the appropriate local permissions (usually root) or could be performed via `dced` if it stubs a server execution object for itself.

Takes the '-force' option to stop any remianing processes with a `server stop -method hard` command.

(f) `host show`

Returns a list describing all processes that are configured to run on the specified host. The optional argument is the fully qualified DCE name of a host. If not given, the local host is assumed. The returned list is of the form:

(i) Server name as output by `server catalog -simplename`.

(ii) One of the tokens: `running` or `notrunning`.

(iii) An optional server specific comment such as `master` or `replica` for a security server and `clerk` or `server` for a DTS server.

(g) `host unconfigure`

Unconfigures a specified host from a cell. Deletes all objects, directories and links from `/.:/hosts/`*`hostname`* including the directory itself. Deletes all principal names begining with `hosts/`*`hostname/`*, this will automatically remove all accounts of the same names. Removes all local configuration files, and stops all running DCE processes (`dced` last). Takes the fully qualified name of a host to unconfigure as an argument. This command may not be run from the host to be unconfigured.

8.12.3. cell

The `cell` object represents a single DCE cell as a whole. This includes all machines, services, resources, principals, etc. In the initial release the operations on an entire cell will be limited, future expansion could include:

(a) `cell create`

Creates a new cell, the equivalent of `dce_config`'s Initial Cell Configuration menu.

The argument to all of these command is a single cellname (not a list of cellnames). If omitted, `/.:` is the default. Supports the following operations:

(a) `cell show`

Returns attributes describing the configuration of the specified cell. The returned attributes are:

(i) `secservers` — each value is a the name of a security server.

(ii) `cdsservers` — each value is the name of a machine running a CDS server. The name is the simple name found under `/.:/hosts`.

(iii) `dtsservers` — each value is the name of a dts server in the cell.

(iv) `hosts` — each value is the name of a host in the cell, including machines mentioned above as servers. This is really `dir list /.:/hosts -simplename`.

The names returned are fully qualified, unless the `-simplename` option is used:

```
dcecp> cell show
{secservers
 /.../dcecp_cell.osf.org/subsys/dce/sec/master
 /.../dcecp_cell.osf.org/subsys/dce/sec/ice}
{cdsservers
 /.../dcecp_cell.osf.org/hosts/frick}
{dtsservers
 /.../dcecp_cell.osf.org/hosts/frick
 /.../dcecp_cell.osf.org/hosts/ice
 /.../dcecp_cell.osf.org/hosts/ninja}
{hosts
```

```
 /.../dcecp_cell.osf.org/hosts/frick
 /.../dcecp_cell.osf.org/hosts/ice
 /.../dcecp_cell.osf.org/hosts/ninja}
dcecp> cell show -simple
{secservers
 subsys/dce/sec/master
 subsys/dce/sec/ice}
{cdsservers
 hosts/frick}
{dtsservers
 hosts/frick
 hosts/ice
 hosts/ninja}
{hosts
 hosts/frick
 hosts/ice
 hosts/ninja}
```

(b) `cell ping`

Used as quick check to see if a cell is running. Can function in three different ways:

(i) If called with no option it pings (using `server ping`) the master security server, all CDS servers,[39] and all the DTS servers in the cell. In case of failure it generates an error and returns a list of servers that could not be contacted. In case of success it returns `DCE services available.`

(ii) The `-replicas` option will cause the command to ping each security and CDS replica as well as those mentioned above. In case of failure it generates an error and returns a list of servers that could not be contacted. In case of success it returns `DCE servers available.`

(iii) The `-clients` option will cause the command to ping every machine in the cell. It does this by looping though `/.:/hosts` and doing a `host ping` on the hostname. In case of failure it generates an error and returns a list of hosts that could not be contacted. In case of success it returns `DCE clients available.`

(c) `cell backup`

Backs up the master security database and each clearinghouse with master replicas in the cell. Depends on there being a `dced` running on each of the server hosts. Takes no arguments or options.

8.12.4. cellalias

The `cellalias` object represents alias names of the cell. Unlike the `cdsalias` object it's scope is for all DCE core components and not just CDS. It can be used to define a new alias for the cell and for informing the security service, the CDS service, and each host about the new alias name, in that order. It can also be used to set a new primary name for the cell informing each involved service in order. Supports the following operations:

(a) `cellalias create`

Creates a new alias cellname for the local cell. The required argument is a single fully qualified alias name of the cell. If the alias name already exists an error is returned.

This command first informs the security service of the new name and sees that all security servers replicate the update, then it informs cds via the `cdsalias create` command, then each host in the cell via

---

39. We had thought of only pinging those servers that have clearinghouses with master replicas, but there is no way to determine which clearinghouses are masters without contacting each clearinghouse and without possible looking at each directory in the cell.

`dced`'s `hostdata` command. Returns an empty string on success. On failure it returns a list of names (registry names, clearinghouse? names, and hostnames) that could not be contacted. This command tries to complete each of the three phases but will fail at the end of a phase.

> *NOTE: permissions required? At least administer permission on the root directory of the cell. Sec? Hostdata?*

(b) `cellalias set`

Sets the primary name of the cell to the alias named in the required argument. The required argument is a single fully qualified name of an existing cell alias. This command should only be used if the CDS and Security servers as well as each host in the cell know about the cell alias (`cellalias create` will do this). Returns an empty string on success. Errors are handled the same as the `cellalias create` command.

> *NOTE: permissions required? At least administer permission on the root directory of the cell. Sec? Hostdata?*

8.13. Command-line Editing Commands

A line may be edited before it is sent to the `dcecp` by typing control characters or escape sequences. A control character, shown as a ''C-'' followed by a letter, is typed by holding down the ''control'' key while the letter is typed. For example, `C-A` is a control-A. An escape sequence is entered by typing the ''escape'' key followed by one or more characters. The escape key is abbreviated as `ESC`. Note that unlike control keys, case matters in escape sequences; `ESC F` is not the same as `ESC f`.

An editing command may be typed anywhere on the line, not just at the beginning. In addition, a return may also be typed anywhere on the line, not just at the end.

Most editing commands may be given a repeat count, *n*, where *n* is a number. To enter a repeat count, type the escape key, the number, and then the command to execute. For example, `ESC 4 C-f` moves forward four characters. If a command may be given a repeat count then the text ''[n]'' is given at the end of its description.

The following control characters are accepted:

(a)	C-A	Move to the beginning of the line
(b)	C-B	Move left (backwards) [n]
(c)	C-D	Delete character [n]
(d)	C-E	Move to end of line
(e)	C-F	Move right (forwards) [n]
(f)	C-G	Ring the bell
(g)	C-H	Delete character before cursor (backspace key) [n]
(h)	C-I	Complete filename (tab key); see below
(i)	C-J	Done with line (return key)
(j)	C-K	Kill to end of line (or column [n])
(k)	C-L	Redisplay line
(l)	C-M	Done with line (alternate return key)
(m)	C-N	Get next line from history [n]
(n)	C-P	Get previous line from history [n]
(o)	C-R	Search backward (forward if [n]) through history for text; must start line if text begins with an uparrow
(p)	C-T	Transpose characters
(q)	C-V	Insert next character, even if it is an edit command
(r)	C-W	Wipe to the mark
(s)	C-XC-X	Exchange current location and mark
(t)	C-Y	Yank back last killed text
(u)	C-[	Start an escape sequence (escape key)
(v)	C-]*c*	Move forward to next character *c*
(w)	C-?	Delete character before cursor (delete key) [n]

The following escape sequences are provided.

(a)	ESC C-H	Delete previous word (backspace key) [n]
(b)	ESC DEL	Delete previous word (delete key) [n]
(c)	ESC SPC	Set the mark (space key); see C-XC-X and C-Y above
(d)	ESC .	Get the last (or [n]'th) word from previous line
(e)	ESC ?	Show possible completions; see below
(f)	ESC <	Move to start of history
(g)	ESC >	Move to end of history
(h)	ESC b	Move backward a word [n]
(i)	ESC d	Delete word under cursor [n]
(j)	ESC f	Move forward a word [n]
(k)	ESC l	Make word lowercase [n]
(l)	ESC u	Make word uppercase [n]
(m)	ESC y	Yank back last killed text
(n)	ESC w	Make area up to mark yankable
(o)	ESC *nn*	Set repeat count to the number *nn*

There is filename completion. Suppose the root directory has the following files in it:

```
bin     vmunix
core    vmunix.old
```

If you type `rm /v` and then the tab key, as much of the name as possible will be finished off by adding `munix`. Because the name is not unique, it will then beep. If you type the escape key and a question mark, it will display the two choices. If you then type a period and a tab, the filename will be finished for you:

rm /v[TAB]*munix.*[TAB]*old*

The tab key is shown by `[TAB]` and the automatically entered text is shown in italics.

9. API'S

All commands in `dcecp` will be implemented as Tcl commands. Therefore, they are user-visible and can be used in `dcecp` scripts to implement new commands. Some commands are implemented in C since they make DCE library calls; they are also visible to users. However, often there will be another command written in Tcl that will wrap the C command to present a better interface to the user. In this case both commands are available to the user, but only the higher-level command will be documented.

The lower-level commands can be found in the Tcl source, and can be considered in some sense to be `dcecp` API's. They will not typically be called directly by users, but will be visible to the ''Tcl-knowledgeable'' (i.e., they will be listed if an `info commands` is performed).

For example, there will be one `create` command implemented in Tcl, it will read its first argument to determine the type of object being created (e.g., a group, a principal, a directory, etc.); it will then call another routine (usually implemented in C) for the lower-level functionality. This lower-level routine will be named the same as the upper-level command with an underscore and the type of object appended to it (e.g., `create_group`, `create_principal`, `create_directory`, etc.).

10. REMOTE INTERFACES

Not applicable.

11. MANAGEMENT INTERFACES

Not applicable.

12. RESTRICTIONS AND LIMITATIONS

Not applicable.

13. OTHER COMPONENT DEPENDENCIES

The `dcecp` will be dependent on the following DCE components:

(a) The NSI.

(b) The CDSPI.

(c) The `sec_rgy` interfaces.

(d) The `sec_id` interfaces.

(e) The `sec_acl` interfaces.

(f) The `sec_login` interfaces.

(g) The DTS interfaces.

(h) The Serviceability interfaces.

(i) The `dced` interfaces.

(j) The Auditing interfaces.

(k) The `editline` functionality from `libdceutils.a`.

(l) The `sec_rgy_attr` interfaces and when written the `sec_attr` interfaces.

14. COMPATIBILITY

The existing control programs will still be released, `dcecp` will be released in addition to them. As this is the first release of `dcecp` there is nothing for it to be compatible with. It makes use of DCE API's, so as long as they remain compatible with older versions of DCE servers, `dcecp` should be able to contact pre-1.1 DCE services.

15. STANDARDS

Not applicable except in that the `dcecp` code will follow the DCE Coding Style Guide [RFC 34].

16. OPEN ISSUES

None.

APPENDIX A. MAPPING OF EXISTING CONTROL PROGRAM COMMANDS

In the mappings lists of this Appendix, the commands in the left column map to the commands in the right column, unless specified otherwise.

A.1. rpccp

These are commands that are mapped directly:

(a)	`add element`	`rpcprofile add`
(b)	`add entry`	`rpcentry create`
(c)	`add member`	`rpcgroup add`
(d)	`export`	`rpcentry export`
(e)	`import`	`rpc-{entry,group,profile} \` `import`
(f)	`remove element`	`rpcprofile remove`
(g)	`remove entry`	`rpcentry delete`
(h)	`remove group`	`rpcgroup delete`
(i)	`remove mapping`	`endpoint delete`
(j)	`remove member`	`rpcgroup remove`
(k)	`remove profile`	`rpcprofile delete`
(l)	`show entry`	`rpcentry show`

(m)	`show group`	`rpcgroup list`
(n)	`show mapping`	`endpoint show -all`
(o)	`show profile`	`rpcprofile show`
(p)	`show server`	`rpcentry show`
(q)	`unexport`	`rpcentry unexport`

A.2. cdscp

These are commands that are mapped directly:

(a)	`add directory`	`directory modify`
(b)	`add object`	`object modify`
(c)	`clear cached server`	`cdscache delete`
(d)	`clear clearinghouse`	`clearinghouse disable`
(e)	`create child`	`directory add`
(f)	`create clearinghouse`	`clearinghouse create`
(g)	`create directory`	`directory create`
(h)	`create link`	`link create`
(i)	`create object`	`object create`
(j)	`create replica`	`directory create -replica\ -clearinghouse`
(k)	`define cached server`	`cdscache create`
(l)	`delete child`	`directory remove`
(m)	`delete clearinghouse`	`clearinghouse delete`
(n)	`delete directory`	`directory delete`
(o)	`delete link`	`link delete`
(p)	`delete object`	`object delete`
(q)	`delete replica`	`directory delete -replica\ -clear`
(r)	`dump clerk cache`	`cdscache dump`
(s)	`list child`	`directory list`
(t)	`list clearinghouse`	`clearinghouse catalog`
(u)	`list directory`	`directory list`
(v)	`list link`	`directory list`
(w)	`list object`	`directory list`
(x)	`remove directory`	`directory modify`
(y)	`remove link`	`link modify`
(z)	`remove object`	`object modify`
(aa)	`set cdscp preferred \ clearinghouse`	`set _s(cds)`
(ab)	`set directory`	`directory modify`
(ac)	`set directory to skulk`	`directory synchronize`
(ad)	`set link`	`link modify`
(ae)	`set object`	`object modify`
(af)	`show cached clearinghouse`	`cdscache show -clear`
(ag)	`show cached server`	`cdscache show -server`
(ah)	`show cdscp preferred \ clearinghouse`	`set _s(cds)`
(ai)	`show child`	`directory show -child`
(aj)	`show clearinghouse`	`clearinghouse show`
(ak)	`show directory`	`directory show`
(al)	`show link`	`link show`
(am)	`show object`	`object show`
(an)	`show replica`	`directory show -replica -clear`

These are commands which don't have a mapping:

(a) `disable clerk`
(b) `disable server`
(c) `set directory to new epoch`
(d) `show cdscp confidence`
(e) `show cell`
(f) `show clerk`
(g) `show server`

A.3. acl_edit

These are commands that are mapped directly:

(a)	`assign`	`acl replace -acl [cat` *filename*`]`[40]
(b)	`cell`	`acl modify -cell`
(c)	`delete`	`acl modify -remove`
(d)	`get_access`	`acl check`
(e)	`kill_entries`	`acl delete`
(f)	`list`	`acl show`
(g)	`modify`	`acl modify {-add	-change}`
(h)	`permissions`	`acl permissions`
(i)	`purge`	`acl purge`
(j)	`substitute`	`acl replace`

These are commands which don't have a mapping:

(a)	`abort`	not applicable
(b)	`commit`	not applicable, use `RET`
(c)	`test_access`	use `acl check` instead

A.4. rgy_edit

A.4.1. Principal, group, and organization commands

These are commands that are mapped directly. In the following list *pgo* is used to mean one of: `principal`, `group`, or `organization`. Also, *go* is used to mean one of: `group` or `organization`.

(a)	`add`	*pgo* `create`
(b)	`change`	*pgo* `modify`
(c)	`delete`	*pgo* `delete`
(d)	`member`	*go* `[add	remove]`
(e)	`view`	*pgo* `show/catalog`
(f)	`adopt`	*pgo* `create -uuid`

A.4.2. Account commands

These are commands that are mapped directly:

(a)	`add`	`account create`
(b)	`cell`	`registry connect`
(c)	`change`	`account modify`
(d)	`delete`	`account delete`
(e)	`view`	`account show/catalog`

---

40. The `assign` command has the ability to set the default cell of the ACL, this can be done using the `-cell` option on the `acl replace` command.

A.4.3. Key management commands

These are commands that are mapped directly (via a new `dced` related `dcecp` command):

(a) `ktadd`	`keytab create`
(b) `ktlist`	`keytab show`
(c) `ktdelete`	`keytab delete`

A.4.4. Miscellaneous commands

These are commands that are mapped directly:

(a) `auth_policy`	`registry modify/show -policies`
(b) `login`	`login`
(c) `policy`	`registry modify/show -policies`
(d) `properties`	`registry modify`

These are commands which don't have a mapping:

(a) `defaults`	not needed, use scripts
(b) `domain`	not needed, use separate objects
(c) `scope`	not needed, use scripts

A.4.5. Local registry commands

These are commands which don't have a mapping:

(a) `delete`
(b) `purge`
(c) `view`

A.5. dtscp

These are commands that are mapped directly:

(a) `advertise`	`dts configure`
(b) `create`	now an option on `dtsd`[41]
(c) `change`	`clock set -abruptly`
(d) `delete`	`dts stop`
(e) `disable`	`dts deactivate`
(f) `enable`	`dts activate`
(g) `set`	`dts modify`
(h) `show`	`dts/clock show`
(i) `synchronize`	`dts/clock synchronize`
(j) `unadvertise`	`dts configure`
(k) `update`	`clock set`

A.6. sec_admin

These are commands that are mapped directly:

(a) `become -master`	`registry set -master`
(b) `become -slave`	`registry set -slave`
(c) `change_master`	`registry set`
(d) `delrep`	`registry delete`
(e) `delrep -force`	`registry delete -force`
(f) `destroy`	`registry delete -only`

---

41. See DCE CR 8353.

(g) `info`	`registry show -replica`
(h) `initrep`	`registry synchronize`
(i) `lrep`	`registry catalog`
(j) `lrep -prop`	`registry show -master`
(k) `lrep -uuid`	`registry show -replica`
(l) `lrep -state`	`registry show -replica`
(m) `lrep -addr`	`registry show -replica`
(n) `lrep -all`	`registry dump`
(o) `master_key`	`registry modify -key`
(p) `site`	`set _s(sec)`
(q) `state`	`registry enable/disable`
(r) `stop`	`registry stop`

These are commands which don't have a mapping:

(a) `monitor`	use script with loop and POSIX sleep

APPENDIX B. LIST OF SECTION TITLES

12. RESTRICTIONS AND LIMITATIONS

13. OTHER COMPONENT DEPENDENCIES

14. COMPATIBILITY

15. STANDARDS

16. OPEN ISSUES

APPENDIX C. TCL COMMANDS

Below is a list of generic Tcl commands for which the implementation depends merely on ANSI C. Task scripts will assume that all of these are implmented in dcecp:

(a) `append`
(b) `array`
(c) `break`
(d) `case`
(e) `catch`
(f) `concat`
(g) `continue`
(h) `error`
(i) `eval`
(j) `expr`
(k) `for`
(l) `foreach`
(m) `format`
(n) `global`
(o) `history`
(p) `if`
(q) `incr`
(r) `info`
(s) `join`
(t) `lappend`
(u) `lindex`
(v) `linsert`
(w) `list`
(x) `llength`
(y) `lrange`
(z) `lreplace`
(aa) `lsearch`
(ab) `lsort`
(ac) `proc`
(ad) `regexp`
(ae) `regsub`
(af) `rename`
(ag) `return`
(ah) `scan`
(ai) `set`
(aj) `split`
(ak) `string`
(al) `switch`
(am) `trace`
(an) `unset`

(ao) `uplevel`
(ap) `upvar`
(aq) `while`

These are the Tcl commands for whom the implementation depends on Unix System Calls. Task script implementation will try to avoid use of these if possible and will document any use of these in the porting guide.

(a) `cd`
(b) `close`
(c) `eof`
(d) `exec`
(e) `exit`
(f) `file`
(g) `flush`
(h) `gets`
(i) `glob`
(j) `open`
(k) `pid`
(l) `puts`
(m) `pwd`
(n) `read`
(o) `seek`
(p) `source`
(q) `tell`
(r) `time`

REFERENCES

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

[PORTG] DCE User's Guide and Reference, Open Software Foundation, 1992.

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

[POSIX.6] IEEE Std 1003.6 — 1991, Portable Operating System Interface — Security Interface, Draft 12. and Utilities.

[RFC 23] R. Mackey, ‘‘DCE 1.1 Internationalization Guide’’, January 1993.

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

[RFC 28] S. Luan, R. Weisz, ‘‘A DCE Server Auditable-Event Identification and a Proposed Audit Logging API’’, December 1992.

[RFC 29] S. Luan, R. Weisz, ‘‘A Design of the DCE Audit Subsystem — An Implementation Specification’’, December 1992.

[RFC 30] H. Melman, ‘‘DCE Administrative Improvement: Requirements for DCE Shell’’, December 1992.

[RFC 34] H. Melman, ‘‘DCE 1.1 Coding Style Guide’’, April 1993.

[RFC 40] S. Martin, ‘‘OSF Character and Code Set Registry’’, April 1993.

[RFC 47.1] R. Salz, P. Wang, ‘‘DCED: The DCE Host Daemon — Command Set’’, January 1994.

[RFC 47.3] R. Salz, P. Wang, ‘‘DCED: The DCE Host Daemon — Functional Specification’’, April 1994.

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

[TCL MAN] J. Ousterhout, ‘‘tcl(3) — overview of tool command language facilities’’, unpublished manual page, University of California at Berkeley, December 1992.

[TCL BOOK] J. Ousterhout, ''Tcl and the Tk Toolkit'', unpublished draft of book to be published by Addison-Wesley in early 1994.

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