11.2.5 Extended Example

As an extended example, suppose we wanted to set up an ACL for the following situation:

1. Anybody can read most attributes.
2. Nobody can read password except the user.
3. User can modify homeTelephone number, everybody else can read it.
4. Sys Admin can modify workTelephoneNumber and PayrollNumber, everybody else can read them.
5. Only user and Sys Admin can read the PayrollNumber.

The first stage is to consider the requirements of various groups of users, so we could write:

```
acl= group  # c=GB@o=UCL@cn=Admin # write # entry
acl= group  # c=GB@o=UCL@cn=Admin # write # \
            attributes # workTelephone $ PayrollNumber
```

But later on in the ACL definition we will want different ACLs for certain attributes, so we need to split into two different groups at this stage.

```
acl= group  # c=GB@o=UCL@cn=Admin # write # entry
acl= group  # c=GB@o=UCL@cn=Admin # write # \
            attributes # workTelephone
acl= group  # c=GB@o=UCL@cn=Admin # write # \
            attributes # PayrollNumber
```

If you do not do this, the DSA will complain about inconsistent ACL definitions, as it will be unable to determine which ACL line to use for which attribute.

We need to give "self" access to various parts of the entry. We do not need to mention workPhone as the access given to "others" is sufficient. Again we split the attribute definitions into small groups.

```
acl= self   # write # entry
acl= self   # write # default
acl= self   # write # attributes # userpassword
acl= self   # write # attributes # homeTelephone
acl= self   # read  # attributes # PayrollNumber
```

Finally, we need to give others access to required attributes.

```
acl= others # read # entry
acl= others # read # default
acl= others # compare # attributes # userpassword
acl= others # read # attributes # homeTelephone
acl= others # read # attributes # workTelephone
acl= others # none # attributes # PayrollNumber
```

```
ACLInfo ::= SET OF SEQUENCE {
        AccessSelector,
        AccessCategories }

AccessCategories ::= ENUMERATED {
        none (0),
        detect (1),
        compare (2),
        read (3),
        add (4),
        write (5) }

AccessSelector ::= CHOICE {
        entry [0] NULL,
                -- DUA identified by the entry
        other [2] NULL,
                -- This indicates "public" rights
        prefix [3] NameList,
                -- This identifies a prefix name for specified DUAs
                -- e.g., anyone in the UK
        group [4] NameList
                -- For specifying group rights
        }

NameList ::= SET OF DistinguishedName

ACLSyntax ::= SEQUENCE {
    childACL            [0] ACLInfo DEFAULT {{other, read}},
    entryACL            [1] ACLInfo DEFAULT {{other, read}},
    defaultAttributeACL [2] ACLInfo DEFAULT {{other, read}},
    [3] SET OF AttributeACL }
                        -- Defaults to a publicly readable
                        -- read only directory

AttributeACL ::= SEQUENCE {
    SET OF AttributeType,
    ACLInfo }

ACL ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX ACLSyntax
    SINGLE VALUE
```

Figure 11.1: ACL Definition

Part III

Administrator's Guide

Chapter 12

Installing QUIPU

This section describes how to install QUIPU, and make it operate in a basic fashion. This is reasonably prescriptive, as it should be possible to install and operate a QUIPU DUA and/or DSA without too much knowledge about how it functions.

QUIPU comes in various separate parts of the ISODE source tree. Only the *libdsap*(3n) library, the DSA *ros.quipu* and the DUA interface DISH are "made" as part of the default installation of ISODE. This section assumes you have installed this part of ISODE. You should consult the *READ-ME* file in the top level of the source tree to find out how to do this. Your attention is drawn to the discussion of the *iaed* and *dased* programs in this *READ-ME* file. This can be used to replace the *isoentities*(5n) and *isoservices*(5n) static files with dynamic directory lookup[1]. *Volume One* to *Volume Four* of this manual describe other features of this installation of ISODE not specific to the OSI directory.

Before you install QUIPU there are various compile time options you could consider setting which control the operation of QUIPU, in particular of the DSA. These options are set in the file *h/quipu/config.h*, and are described in the next section. If you consider yourself a QUIPU "novice", then these are probably best left to their default values initially. However, if you qualify as a "large" site you may want to consider enabling the TURBO_DISK options.

In the *others/quipu/uips/* directory of the ISODE source tree there are various sub-directories, one for each optional user interface (FRED, SD, POD

[1]In fact, you are encouraged to use these services rather that the static (potentially out of date) files.

and DE). A version of DISH that runs directly from a UNIX shell, *dishinit*; a script to create a default *.quipurc* file for new users, and *sid*: a set of scripts that utilise the shell version of DISH, are all installed from the *dish* sub-directory. The *manage* sub-directory contains an enhanced version of DISH that can be used to manage alias attributes. You should consult *others/quipu/uips/READ-ME*, for precise installation details of all of these interfaces.

Each of these interfaces knows about the `photo` attribute that an entry in the DIT can have. In order to display the photographs, the photo handling code must be compiled. Instructions on this can be found in Section 14.7.3 of this manual and the file *others/quipu/photo/READ-ME*.

12.1 Compile Options

This section describes the options that can be set in the *h/quipu/config.h* before compilation of the QUIPU code.

`#define PDU_DUMP`: If this is defined, and DISH is invoked with

```
dish -pdu foobar
```

then a directory *foobar* will be created, and will contain logs of all the X.500 PDUs sent to and from the DSA. This is useful for debugging.

`#define NO_STATS`: If defined, the QUIPU will *not* produce statistical and audit logs either the DSA or DUA. These logs are useful to see what has been happening to your system. If logging is allowed it can be turned off at runtime. From the standpoint of security, it is advisable not to select this option. Audit logs are very useful for detecting and tracing attempts to break the security of the system.

`#define CHECK_FILE_ATTRIBUTES`: If an EDB entry contains a FILE attribute, check that the corresponding file exists at load time. This significantly increases the time taken to start a DSA.

`#define QUIPU_MALLOC`: Use a version of malloc() optimised for the memory resident QUIPU DSA database. It is believed to behave

about 20% faster than the standard malloc algorithm in QUIPU's case.

`#define TURBO_DISK`: This option is described in the next section of this manual.

`#define TURBO_INDEX`: This option is described in the next section of this manual.

`#define SOUNDEX_PREFIX`: Consider soundex prefixes as matches. For example, make "fred" match "frederick". Defining this option gives approximate matching behavior the same as in QUIPU-6.0.

`#define COMPAT_6_0`: Operate in a manner compatible with QUIPU-6.0. For operations involving subtree searching across multiple DSA this option is required when one of the DSAs is based on QUIPU-6.0. It is *not* required for QUIPU-6.1 and above.

`#define USE_BUILTIN_OIDS`: A DSA needs to know the OIDs for various attribute type and object classes. With this option, the OIDs are built into the code (for efficiency, and to remove the table dependency), without it the *oidtables* are used.

For the DISH interface, a compile option can be set to allow the use of the GNU *readline* package from the Free Software Foundation. Full details are given in the *quipu/dish/Makefile*.

12.2 TURBO Options for Large Sites

This section describes two options (TURBO_INDEX and TURBO_DISK) that can be set in the file *h/quipu/config.h*. Both options are most useful for sites with a large amount of data (e.g., thousands of entries, or hundreds of EDB files). The TURBO_INDEX option is set by default. If your DSA only holds a small amount of data you can skip this section. [2]

The first option, TURBO_INDEX, can be used to speed certain kinds of searches by building an index based on selected attribute types. There

[2]The TURBO_AVL and TURBO_LOAD options available in previous releases of QUIPU are no longer required.

are three DSA tailor file options that are used to specify which attributes and which portions of the tree are to be indexed. They are `optimize_attr`, `index_subtree`, and `index_siblings`. Section 14.3 describes these options in more detail. For the selected attributes and portions of the tree, the index is searched for queries involving equality, approximate equality, initial substring matching, and attribute existence. *and* and *or* combinations of the above queries are also supported. Queries involving negation, inequality, or substring matching with no initial substring supplied are not indexed and will be handled as usual, by a linear search of the database.

The second option, TURBO_DISK, can be used to make modify operations much faster, especially on large data sets. It requires the use of the *gdbm* library. *Gdbm* is a library of simple database routines providing functionality similar to *dbm* and *ndbm*, but without the filesystem page size limitations of those systems. *Gdbm* is GNU software and is available from the Free Software Foundation. It is not a part of the ISODE.

Normally, when an entry is modified, QUIPU writes the entire EDB file containing the entry out to disk. If the EDB file is very large, this can take some time, especially on a heavily loaded system. The TURBO_DISK option works by keeping the disk data in *gdbm* files instead of the regular EDB files. This way, when an update is made, only the affected entry need be written. The performance increase is directly proportional to the size of the EDB file. Whereas a normal update operation takes time proportional to the size of the EDB file, with the TURBO_DISK option it takes a small constant amount of time. If you don't have EDB files with at least several hundred entries, it's probably not worth enabling the TURBO_DISK option.

To use the TURBO_DISK option, you should add the following define to the *h/quipu/config.h* file:

```
#define TURBO_DISK /* enable fast EDB update operations */
```

You should also add the following to the *config/CONFIG.make* file:

```
LIBGDBM= -lgdbm
```

If you have defined TURBO_DISK, you will have to convert your EDB file hierarchy into a gdbm file hierarchy before running QUIPU. This step is only necessary once. The *quipu/turbo* directory contains some tools to help in this process. The shell script *tree2dbm* will convert an EDB file hierarchy to