The ISO Development Environment: User's Manual

Volume 5: QUIPU

Colin J. Robbins
X-Tel Services Ltd

Stephen E. Kille
Department of Computer Science
University College London

Tue Mar 26 20:20:56 EET 1991
`Draft Version #6.15`

Contents

[0] This document (version #6.15) was LaTeXset March 26, 1991 with lplain v2.09-10/29/85.

List of Tables

List of Figures

Preface

The software described herein has been developed as a research tool and represents an effort to promote the use of the International Organization for Standardization (ISO) interpretation of Open Systems Interconnection (OSI), particularly in the Internet and RARE research communities.

Notice, Disclaimer, and Conditions of Use

The ISODE is openly available but is **NOT** in the public domain. You are allowed and encouraged to take this software and build commercial products. However, as a condition of use, you are required to "hold harmless" all contributors.
Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that this notice and the reference to this notice appearing in each software module be retained unaltered, and that the name of any contributors shall not be used in advertising or publicity pertaining to distribution of the software without specific written prior permission. No contributor makes any representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

ALL CONTRIBUTORS DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS. IN NO EVENT SHALL ANY CONTRIBUTOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

As used above, "contributor" includes, but is not limited to:

The MITRE Corporation
The Northrop Corporation
NYSERNet, Inc.
Performance Systems International, Inc.
University College London
The University of Nottingham
X-Tel Services Ltd
The Wollongong Group, Inc.
Marshall T. Rose

In particular, the Northrop Corporation provided the initial sponsorship for the ISODE and the Wollongong Group, Inc., also supported this effort. The

ISODE receives partial support from the U.S. Defense Advanced Research Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602–88–C–0016 to NYSER-Net Inc.

Revision Information

This document (version #6.15) and its companion volumes are believed to accurately reflect release v 6.0 of March 26, 1991.

Release Information

If you'd like a copy of the release described in this document, there are several avenues available:

- NORTH AMERICA
  For mailings in NORTH AMERICA, send a check for 375 US Dollars to:

Postal address:	University of Pennsylvania Department of Computer and Information Science Moore School Attn: David J. Farber (ISODE Distribution) 200 South 33rd Street Philadelphia, PA 19104-6314 US
Telephone:	+1 215 898 8560

  Specify one of:

  1. 1600bpi 1/2–inch tape, or
  2. Sun 1/4–inch cartridge tape.

  The tape will be written in *tar* format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- EUROPE
  For mailings in EUROPE, send a cheque or bankers draft and a purchase order for 200 Pounds Sterling to:

Postal address:	Department of Computer Science Attn: Natalie May/Dawn Bailey University College London Gower Street London, WC1E 6BT UK

For information only:

Telephone:	+44 71 380 7214
Fax:	+44 71 387 1397
Telex:	28722
Internet:	`natalie@cs.ucl.ac.uk`
	`dawn@cs.ucl.ac.uk`

Specify one of:

1. 1600bpi 1/2–inch tape, or
2. Sun 1/4–inch cartridge tape.

The tape will be written in *tar* format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- EUROPE (tape only)
  Tapes without hardcopy documentation can be obtained via the European UNIX[1] User Group (EUUG). The ISODE 6.0 distribution is called EUUGD14.

Postal address:	EUUG Distributions
	c/o Frank Kuiper
	Centrum voor Wiskunde en Informatica
	Kruislann 413
	1098 SJ Amsterdam
	The Netherlands
For information only:	
Telephone:	+31 20 5924056
	(or +31 20 5929333)
Telex:	12571 mactr nl
Telefax:	+31 20 5924199
Internet:	`euug-tapes@cwi.nl`

Specify one of:

[1] UNIX is a trademark of AT&T Bell Laboratories.

1. 1600bpi 1/2–inch tape: 130 Dutch Guilders
2. 800bpi 1/2–inch tape: 130 Dutch Guilders
3. Sun 1/4–inch cartridge tape (QIC-24 format): 190 Dutch Guilders
4. 1600 1/2–inch tape (QIC-11 format): 190 Dutch Guilders

If you require DHL this is possible and will be billed through. Note that if you are not a member of EUUG, then there is an additional handling fee of 300 Dutch Guilders (please encloses a copy of your membership or contribution payment form when ordering). Do not send money, cheques, tapes or envelopes, you will be invoiced.

- PACIFIC RIM
For mailings in the Pacific Rim, send a cheque for 250 dollars Australian to:

Postal address:	CSIRO DIT Attn: Andrew Waugh (ISODE Distribution) 55 Barry Street Carlton, 3053 Australia

For information only:

Telephone:	+61 3 347 8644
Fax:	+61 3 347 8987
Internet:	`ajw@ditmela.oz.au`

Specify one of:

1. 1600/3200/6250bpi 1/2–inch tape, or
2. Sun 1/4—inch cartridge tape in either QIC-11 or QIC-24 format.

The tape will be written in tar format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- Internet
If you can FTP to the Internet, you can use anonymous FTP to the host

uu.psi.com [136.161.128.3] to retrieve isode-6.tar.Z in BINARY mode from the isode/ directory. This file is the *tar* image after being run through the compress program and is approximately 4.5MB in size.

- NIFTP
  If you run NIFTP over the public X.25 or over JANET, and are registered in the NRS at Salford, you can use NIFTP with username "guest" and your own name as password, to access UK.AC.UCL.CS to retrieve the file <SRC>isode-6.tar. This is a 14MB *tar* image. The file <SRC>isode-6.tar.Z is the *tar* image after being run through the compress program (4.5MB).

- FTAM on the JANET or PSS
  The source code is available by FTAM at the University College London over X.25 using JANET (DTE 00000511160013) or PSS (DTE 23421920030013) with TSEL 259 (ASCII encoding). Use the "anon" user-identity and retrieve the file <SRC>isode-6.tar. This is a 14MB *tar* image. The file <SRC>isode-6.tar.Z is the *tar* image after being run through the compress program (4.5MB).

- FTAM on the Internet
  The source code is available by FTAM over the Internet at host osi.nyser.net [192.33.4.10] (TCP port 102 selects the OSI transport service) with TSEL 259 (numeric encoding). Use the "anon" user-identity, supply any password, and retrieve isode-6.tar.Z from the isode/ directory. This file is the *tar* image after being run through the compress program and is approximately 4.5MB in size.

For distributions via FTAM, the file service is provided by the FTAM implementation in ISODE 5.0 or later (IS FTAM).

For distributions via either FTAM or FTP, there is an additional file available for retrieval, called isode-ps.tar.Z which is a compressed tar image (7MB) containing the entire documentation set in PostScript format.

Discussion Groups

The Internet discussion group `ISODE@NIC.DDN.MIL` is used as a forum to discuss ISODE. Contact the Internet mailbox `ISODE-Request@NIC.DDN.MIL` to be asked to be added to this list.

Acknowledgements

Many people have made comments about and contributions to the ISODE which have been most helpful. The following list is by no means complete:

The first three releases of the ISODE were developed at the Northrop Research and Technology Center, and the first version of this manual is referenced as NRTC Technical Paper #702. The initial work was supported in part by Northrop's Independent Research and Development program.

The Wollongong Group supported ISODE for its 4.0 and 5.0 release. they deserve much credit for that. Further, they contributed an implementation of RFC1085, a lightweight presentation protocol for TCP/IP-based internets.

The ISODE is currently supported by Performance Systems International, Inc. and NYSERNet, Inc. It should be noted that PSI/NYSERNet support for the ISODE represents a substantial increase in commitment. That is, the ISODE is now a funded project, whereas before ISODE was always an after-hours activity. The NYSERNet effort is partially support by the U.S. Defense Advanced Research Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602–88–C–0016 to NYSERNet Inc.

Christopher W. Moore of the Wollongong Group has provided much help with ISODE both in terms of policy and implementational matters. He also performed Directory interoperability testing against a different implementation of the OSI Directory.

Dwight E. Cass of the Northrop Research and Technology Center was one of the original architects of *The ISO Development Environment*. His work was critical for the original proof of concept and should not be forgotten. John L. Romine also of the Northrop Research and Technology Center provided many fine comments concerning the presentation of the material herein. This resulted in a much more readable manuscript. Stephen H. Willson, also of the Northrop Research and Technology Center, provided some help in verifying the operation of the software on a system running the AT&T variant of UNIX.

The *librosap*(3n) library was heavily influenced by an earlier native-TCP version written by George Michaelson formerly of University College London, in the United Kingdom. Stephen E. Kille, of University College London, provided valuable feedback on the *pepy*(1) utility. In addition, both Steve and George provided us with some good comments concerning the *libpsap*(3)

library. Steve is also the conceptual architect for the addressing scheme used in the software, and he modified the *librosap*(3n) library to support half-duplex mode when providing ECMA ROS service. George contributed the CAMTEC X.25 interface. Simon Walton, also of University College London, has been very helpful in providing constant feedback on the ISODE during beta-testing.

The INCA project donated the QUIPU Directory implementation to the ISODE. Stephen E. Kille, Colin J. Robbins, and Alan Turland, at the time all of University College London, are the three principals who developed the 6.0 version of the directory software. In addition, Steve Titcombe, also of UCL spent considerable time on the DIrectory SHell (DISH), and Mike Roe formerly of UCL, put a large amount of effort into the security requirements of QUIPU. Development of the current version of QUIPU has been coordinated by Colin J. Robbins now of X-Tel Services Ltd, and designed by Stephen E. Kille.

The UCL work has been partially supported by the commission of the EEC under its ESPRIT program, as a stage in the promotion of OSI standards. Their support has been vital to the UCL activity. In addition, QUIPU is also funded by the UK Joint Network Team (JNT).

Julian P. Onions, of X-Tel Services Ltd is the current *pepy*(1) guru, having brainstormed and implemented the encoding functionality along with Stephen E. Easterbrook formerly of University College London. Julian also contributed the UBC X.25 interface along with the TCP/X.25 TP0 bridge, and has also contributed greatly to *posy*(1). Julian's latest contribution has been a *transport service bridge*. This is used to masterfully solve interworking problems between different OSI stacks (TP0/X.25, TP4/CLNP, RFC1006/TCP, and so on).

John Pavel and Godfrey Cowin of the Department of Trade and Industry/National Physical Laboratory in the United Kingdom both contributed significant comments during beta-testing. In particular, John gave us a lot of feedback on *pepy*(1) and on the early FTAM DIS implementation. John also contributed the SunLink X.25 interface.

Keith Ruttle of CAMTEC Electronics Limited in the United Kingdom contributed the both the driver for the new CAMTEC X.25 interface and the CAMTEC CONS interface (X.25 over 802 networks). This latter driver was later removed from the distribution for lack of use.

In addition, Andrew Worsley of the Department of Computer Science at the University of Melbourne in Australia pointed out several problems with the FTAM DIS implementation. He also developed a replacement for *pepy* and *posy* called *pepsy*. After moving to University College London, he improved this system and integrated into the ISODE.

Olivier Dubous of BIM sa in Belgium contributed some fixes to concurrency control in the FTAM initiator to allow better interworking with the VMS[2] implementation of the filestore. He also suggested some changes to allow interworking with the FTAM T1 and A/111 profiles.

Olli Finni of Nokia Telecommunications provided several fixes found when interoperability testing with the TOSI implementation of FTAM.

Mark R. Horton of AT&T Bell Laboratories also provided some help in verifying the operation of the software on a 3B2 system running UNIX System V release 2. In addition, Greg Lavender of NetWorks One under contract to the U.S. Navy Regional Data Automation Center (NARDAC), provided modifications to allow the software to run on a generic port of UNIX System V release 3.

Steve D. Miller of the University of Maryland provided several fixes to make the software run better on the ULTRIX[3] variant of UNIX.

Jem Taylor of the Computer Science Department at the University of Glasgow provided some comments on the documentation.

Hans-Werner Braun of the University of Michigan provided the inspiration for the initial part of Section 1.2.

A previous release of the software contained an ISO TP4/CLNP package derived from a public-domain implementation developed by the National Institute of Standards and Technology (then called the National Bureau of Standards). The purpose of including the NIST package (and associated support) was to give an example of how one would interface the code to a "generic" TP4 implementation. As the software has now been interfaced to various native TP4 implementations, the NIST package is no longer present in the distribution.

John A. Scott of the MITRE Corporation contributed the SunLink OSI interface for TP4. He also wrote the FTAM/FTP gateway which the MITRE Corporation has generously donated to this package.

Philip B. Jelfs of the Wollongong Group upgraded the FTAM/FTP gate-

[2] VMS is a trademark of Digital Equipment Corporation.
[3] ULTRIX is a trademark of Digital Equipment Corporation.

way to the “IS-level” (International Standard) FTAM.

Rick Wilder and Don Chirieleison of the MITRE Corporation contributed the VT implementation which the MITRE Corporation has generously donated to this package.

Jacob Rekhter of the T. J. Watson Research Center, IBM Corporation provided some suggestions as to how the system should be ported to the IBM RT/PC running either AIX or 4.3BSD. He also fixed the incompatibilities of the FTAM/FTP gateway when running on 4.3BSD systems.

Ashar Aziz and Peter Vanderbilt, both of of Sun Microsystems Inc., provided some very useful information on modifying the SunLink OSI interface for TP4.

Later on, elements of the SunNet 7.0 Development Team (Hemma Prafullchandra, Raj Srinivasan, Daniel Weller, and Erik Nordmark) made numerous enhancements and fixes to the system.

John Brezak of Apollo Computer, Incorporated ported the ISODE to the Apollo workstation. Don Preuss, also of Apollo, contributed several enhancements and minor fixes.

Ole-Jorgen Jacobsen of Advanced Computing Environments provided some suggestions on the presentation of the material herein.

Nandor Horvath of the Computer and Automation Institute of the Hungarian Academy of Sciences while a guest-researcher at the DFN/GMD in Darmstadt, FRG, provided several fixes to the FTAM implementation and documentation.

George Pavlou and Graham Knight of University College London contributed some management instrumentation to the *libtsap*(3n) library.

Juha Heinänen of Tampere University of Technology provided many valuable comments and fixes on the ISODE.

Paul Keogh of the Nixdorf Research and Development Center, in Dublin, Ireland, provided some fixes to the FTAM implementation.

Oliver Wenzel of GMD Berlin contributed the RFA system.

L. McLoughlin of Imperial College contributed Kerberos support for the FTAM responder.

Kevin E. Jordan of CDC provided many enhancements to the G3FAX library.

John A. Reinart of Cray Research contributed many performance enhancements.

Ed Pring of the T. J. Watson Research Center, IBM Corporation provided several fixes to the SMUX implementation in ISODE's SNMP agent.

Finally, James Gosling, author of the superb Emacs screen-editor for UNIX, and Leslie Lamport, author of the excellent LaTeX document preparation system both deserve much praise for such winning software. Of course, the whole crew at U.C. Berkeley also deserves tremendous praise for their wonderful work on their variant of UNIX.

/mtr

Mountain View, California
March, 1991

Part I

Introduction

Chapter 1

Overview

This document describes a non-proprietary implementation of some of the protocols defined by the International Organization for Standardization and International Electrotechnical Commission (ISO/IEC), the International Telegraph and Telephone Consultative Committee (CCITT), and the European Computer Manufacturer Association (ECMA).[1]

The purpose of making this software openly available is to accelerate the process of the development of applications in the OSI protocol suite. Experience indicates that the development of application level protocols takes as long as or significantly longer than the lower level protocols. By producing a non-proprietary implementation of the OSI protocol stack, it is hoped that researchers in the academic, public, and commercial arenas may begin working on applications immediately. Another motivation for this work is to foster the development of OSI protocols both in the European RARE and the U.S. Internet communities. The Internet community is widely known as having pioneered computer-communications since the early 1970's. This community is rich in knowledge in the field, but currently is not actively experimenting with the OSI protocols. By producing an openly available implementation, it is hoped that the OSI protocols will become quickly widespread in the Internet, and that a productive (and *painless*) transition in the Defense Data Network (DDN) might be promoted. The RARE community is the set of corresponding European academic and research organizations. While they do not have the same long implementation experience as the Internet commu-

[1]In the interests of brevity, unless otherwise noted, the term "OSI" is used to denote these parallel protocol suites.

nity, they have a deep commitment to International Standards. It is intended that this release gives vital early access to prototype facilities.

1.1 Fanatics Need Not Read Further

This software can support several different network services below the transport service access point (TSAP). One of these network services is the DoD Transmission Control Protocol (TCP)[JPost81].[2] This permits the development of the higher level protocols in a robust and mature internet environment, while providing us the luxury of not having to recode anything when moving to a network where the OSI Transport Protocol (TP) is used to provide the TSAP. However, the software also operates over pure OSI lower levels of software. it is mainly used in that fashion — outside of the United States.

Of course, there will always be "zealots of the pure faith" making claims to the effect that:

> *TCP/IP is dead! Any work involving TCP/IP simply dilutes the momentum of OSI.*

or, from the opposite end of the spectrum, that

> *The OSI protocols will never work!*

Both of these statements, from diametrically opposing protocol camps are, of course, completely unfounded and largely inflammatory. TCP/IP is here, works well, and enjoys a tremendous base of support. OSI is coming, and will work well, and when it eventually comes of age, it will enjoy an even larger base of support.

The role of ISODE, in this maelstrom that generates much heat and little light, is to provide a useful transition path between the two protocol suites in which complementary efforts can occur. The ISODE approach is to use the strengths of both the DDN and OSI protocol suites in a cooperative and positive manner. For a more detailed exposition of these ideas, kindly refer to [MRose90c] or the earlier work [MRose86].

[2] Although the TCP corresponds most closely to offering a transport service in the OSI model, the TCP is used as a connection-oriented network protocol (i.e., as co-service to X.25).

1.2 The Name of the Game

The name of the software is the ISODE. The official pronunciation of the ISODE, takes four syllables: *I-SO-D-E*. This choice is mandated by fiat, not by usage, in order to avoid undue confusion.

Please, as a courtesy, do not spell ISODE any other way. For example, terms such as ISO/DE or ISO-DE do not refer to the software! Similarly, do not try to spell out ISODE in such a way as to imply an affiliation with the International Organization for Standardization. There is no such relationship. The *ISO* in ISODE is not an acronym for this organization. In fact, the *ISO* in ISODE doesn't really meaning anything at all. It's just a catchy two syllable sound.

1.3 Operating Environments

This release is coded entirely in the *C* programming language[BKern78], and is known to run under the following operating systems (without any kernel modifications):

- Berkeley UNIX
  The software should run on any faithful port of 4.2BSD, 4.3BSD, or 4.4BSD UNIX. Sites have reported the software running: on the Sun-3 workstation running Sun UNIX 4.2 release 3.2 and later; on the Sun Microsystems workstation (Sun-3, Sun-4, and Sun-386i) running SunOS release 4.0 and later; on the VAXstation[3] running ULTRIX, on the Integrated Solutions workstation; and, on the RT/PC running 4.3BSD.[4]

  In addition to using the native TCP facilities of Berkeley UNIX, the software has also be interfaced to versions 4.0 through 6.0 of the SunLink X.25 and OSI packages (although Sun may have to supply you with some modified `sgtty` and `ioctl` include files if you are using an

[3] VAXstation is a trademark of Digital Equipment Corporation.

[4] Do not however, attempt to compile the software with the SunPro *make* program! It is not, contrary to any claims, compatible with the standard *make* facility. Further, note that if you are running a version of SunOS 4.0 prior to release 4.0.3, then you may need to use the *make* program found in */usr/old/*, if the standard *make* you are using is the SunPro *make*. In this case, you will need to put the old, standard *make* in */usr/bin/*, and you can keep the SunPro *make* in */bin/*.

earlier version of SunLink X.25). The optional SunLink Communications Processor running DCP 3.0 software has also been tested with the software.

- AT&T UNIX
  The software should run on any faithful port of SVR2 UNIX or SVR3 UNIX. One of the systems tested was running with an Excelan EXOS[5] 8044 TCP/IP card. The Excelan package implements the networking semantics of the 4.1aBSD UNIX kernel. As a consequence, the software should run on any faithful port of 4.1aBSD UNIX, with only a minor amount of difficultly. As of this writing however, this speculation has not been verified. The particular system used was a Silicon Graphics IRIS workstation.[6]

  Another system was running the WIN TCP/IP networking package. The WIN package implements the networking semantics of the 4.2BSD UNIX kernel. The particular system used was a 3B2 running System V release 2.0.4, with WIN/3B2 version 1.0.

  Another system was also running the WIN TCP/IP networking package but under System V release 3.0. The WIN package on SVR3 systems emulates the networking semantics of the 4.2BSD UNIX kernel but uses STREAMS and TLI to do so.

- AIX
  The software should run on the IBM AIX Operating System which is a UNIX-based derivative of AT&T's System V. The particular system used was a RT/PC system running version 2.1.2 of AIX.

- HP-UX
  The software should run on HP's UNIX-like operating system, HP-UX. The particular system used was an Indigo 9000/840 system running version A.B1.01 of HP-UX. The system has also reported to have run on an HP 9000/350 system under version 6.2 of HP-UX.

[5] EXOS is a trademark of Excelan, Incorporated.

[6] This test was made with an earlier release of this software, and access to an SGI workstation was not available when the current version of the software tested. However, the networking interface is still believed to be correct for the Excelan package.

- ROS
  The software should run on the Ridge Operating system, ROS. The particular system used was a Ridge-32 running version 3.4.1 of ROS.

- Pyramid OsX
  The software should run on a Pyramid computer running OsX. The particular system used was a Pyramid 98xe running version 4.0 of OsX.

Since a Berkeley UNIX system is the primary development platform for ISODE, this documentation is somewhat slanted toward that environment.

1.4 Organization of the Release

A strict layering approach has been taken in the organization of the release. The documentation mimics this relationship approximately: the first two volumes describe, in top-down fashion, the services available at each layer along with the databases used by those services; the third volume describes some applications built using these facilities; the fourth volume describes a facility for building applications based on a programming language, rather that network-based, model; and, the fifth volume describes a complete implementation of the OSI Directory.

In *Volume One*, the "raw" facilities available to applications are described, namely four libraries:

- the *libacsap*(3n) library, which implements the OSI Association Control Service (ACS);

- the *librosap*(3n) library, which implements different styles of the OSI Remote Operations Service (ROS);

- the *librtsap*(3n) library, which implements the OSI Reliable Transfer Service (RTS); and,

- the *libpsap*(3) library, which implements the OSI abstract syntax and transfer mechanisms.

In *Volume Two*, the services upon which the application facilities are built are described, namely three libraries:

- the *libpsap2*(3n) library, which implements the OSI presentation service;
- the *libssap*(3n) library, which implements the OSI session service; and,
- the *libtsap*(3n) library, which implements an OSI transport service access point.

In addition, there is a replacement for the *libpsap2*(3n) library called the *libpsap2-lpp*(3n) library. This implements the lightweight presentation protocol for TCP/IP-based internets as specified in RFC1085.

In addition, *Volume Two* contains information on how to configure the ISODE for your network.

In *Volume Three*, some application programs written using this release are described, including:

- An implementation of the ISO FTAM which runs on Berkeley or AT&T UNIX. FTAM, which stands for File Transfer, Access and Management, is the OSI file service. The implementation provided is fairly complete in the context of the particular file services it offers. It is a minimal implementation in as much as it offers only four core services: transfer of text files, transfer of binary files, directory listings, and file management.
- An implementation of an FTAM/FTP gateway, which runs on Berkeley UNIX.
- An implementation of the ISO VT which runs on Berkeley UNIX. VT, which stands for Virtual Terminal, is the OSI terminal service. The implementation consists of a basic class, TELNET profile implementation.
- An implementation of the "little services" often used for debugging and amusement.
- An implementation of a simple image database service.

In *Volume Four*, a "cooked" interface for applications using remote operations is described, which consists of three programs and a library:

- the *rosy*(1) compiler, which is a stub-generator for specifications of Remote Operations;

- the *posy*(1) compiler, which is a structure-generator for ASN.1 specifications;

- the *pepy*(1) compiler, which reads a specification for an application and produces a program fragment that builds or recognizes the data structures (APDUs in OSI argot) which are communicated by that application; and,

- the *librosy*(3n) library, which is a library for applications using this distributed applications paradigm.

In *Volume Five*, the QUIPU directory is described, which currently consists of several programs and a library:

- the *quipu*(8c) program, which is a Directory System Agent (DSA);

- the *dish*(1c) family of programs, which are a set of DIrectory SHell commands; and,

- the *libdsap*(3n) library, which is a library for applications using the Directory.

1.5 A Note on this Implementation

Although the implementation described herein might form the basis for a production environment in the near future, this release is not represented as as "production software".

However, throughout the development of the software, every effort has been made to employ good software practices which result in efficient code. In particular, the current implementation avoids excessive copying of bytes as data moves between layers. Some rough initial timings of echo and sink entities at the transport and session layers indicate data transfer rates quite competitive with a raw TCP socket (most differences were lost in the noise). The work involved to achieve this efficiency was not demanding.

Additional work was required so that programs utilizing the *libpsap*(3) library could enjoy this level of performance. Although data transfer rates at the reliable transfer and remote operations layers are not as good as raw TCP, they are still quite impressive (on the average, the use of a ROS interface (over presentation, session, and ultimately the TCP) is only 20% slower than a raw TCP interface).

1.6 Changes Since the Last Release

A brief summary of the major changes between v 6.0 and v 6.0 are now presented. These are the user-visible changes only; changes of a strictly internal nature are not discussed.

- A new program, *pepsy*, has been developed to replace both *pepy* and *posy*. It is described in *Volume Four*.
- The *dsabuild* program has been removed, in favor of some shell scripts.
- The "higher performance nameservice" has been discontinued in favor of a "user-friendly nameservice". As such, the syntax of the `str2aei` routine has changed. This routine will soon be deprecated, so get in the habit of using the new `str2aeinfo` routine discussed in *Volume One* on page 15.
- The `na_type` and `na_subnet` fields of the network address structure described in *Volume Two* on page 123 have been renamed. For compatibility, macros are provided. These macros will be removed after this release.
- The stub directory facility is now deprecated in favor of an OSI Directory based approach. As a result, the *aetbuild* program has been removed.

As a rule, the upgrade procedure is a two-step process: first, attempt to compile your code, keeping in mind the changes summary relevant to the code; and, second, once the code successfully compiles, run the code through *lint*(1) with the supplied lint libraries.

Although every attempt has been made to avoid making changes which would affect previously coded applications, in some cases incompatible changes were required in order to achieve a better overall structure.

Chapter 2

Overview of QUIPU

2.1 Summary

QUIPU is a Public Domain implementation of the OSI Directory as specified in CCITT X.500 Recommendations / ISO 9594 for Directory Services [ISO88] [CCITT88]. It is intended to provide an environment for experimentation and for early pilots using Standardised Directory Services. QUIPU is currently aligned to the ISO IS. QUIPU is also aligned to the NIST Directory Implementors Guide Version 1, with the following exceptions:

- QUIPU does not enforce the bounds constraints on attributes, filters or APDU size.
- T.61 string formatting characters are not rejected.
- If a DN is supplied with no password in an unprotected simple bind, QUIPU does not always check to see if the DN exists. If the DSA connected to can say authoritatively the DN does not exist, the association is rejected. However, if a chain operation is required to check the DN, the bind IS allowed.
- When comparing attributes of UTCtime syntax, if the seconds field is omitted, QUIPU does not perform the match correctly (i.e., the seconds field in the attribute values should be ignored, but are not).
- QUIPU always supplies the optional Chaining argument "originator" even if the CommonArgument "requestor" is used.

- QUIPU always supplies the optional Chaining argument "target" even if the base object in the DAP arguments is the same.
- The object class "without an assigned object identifier" is not recognised unless the "alias" object class is also present.
- Non Specific Subordinate References are never followed by a QUIPU DSA, but they are passed on correctly to the client if generated.
- The "entryOnly" chaining argument introduced by the DIG is recognised, but never set by a QUIPU DSA[1]

QUIPU is intended to provide an environment for early experimentation with standardized Directory services. It is used by the ISODE for identification of the location of OSI applications (including QUIPU) and for provision of white and yellow page services. The Directory Abstract Service and DSA Abstract Service defined in [CCITT88, ISO88] and their associated protocols are supported.

Major aspects of the QUIPU implementation are:

- Use of memory structures to provide fast access
- Activity scheduling within the DSA to allow for multiple accesses
- General and flexible searching capabilities
- Extensions to provide access control
- External schema management
- Use of the Directory to control Distributed Operations

The current implementation provides a DSA, and a procedural interface to the Directory Abstract Service, which will enable other applications to use the Directory. There is also a DIrectory SHell interface — DISH. This provides full access to the Directory Abstract Service, using the procedural interface. Standard Distributed Operations are used with both referrals and chaining (using the Directory System Protocol) provided.

A full discussion of the design issues relating to QUIPU can be found in [SKill89b].

[1]It is generated if the COMPAT_6_0 compile option is removed — See Section 12.1.

2.2 Pronouncing QUIPU

The name of the INCA Directory is QUIPU. The official pronunciation of QUIPU takes two syllables: *kwip-ooo.*

2.3 Why QUIPU

QUIPU was originally developed as a part of the INCA project. The Inca of Peru did not have writing. Instead, they stored information on strings, carefully knotted in a specific manner and with coloured thread, and attached to a larger rope. These devices were known as *Quipus.* The encoding was obscure, and could only be read by selected trained people: the *Quipucamayocs.* The Quipu was a key component of Inca society, as it contained information about property and locations throughout the extensive Inca empire.

2.4 Objectives

2.4.1 General Aims

QUIPU has a number of general aims:

- To produce an implementation which follows the emerging OSI Directory standards.
- Flexibility to enable the system to be used for experimentation and research into problems relating to Directory Service.
- Investigation of distribution and replication
- Pilot experimental usage.

2.4.2 Technical Goals

The major goals of the QUIPU Directory Service are:

- Full support of the Directory Access Protocol, Directory System Protocol and Distributed Operations, as defined in [CCITT88].

- Support of the majority of the service elements specified in [CCITT88].
- Ability for interworking with other Directory implementations, including use of referrals and chaining.
- Very full searching and matching capabilities, beyond the minimum required by [CCITT88].

The following are not goals:

- In practice, the memory based approach has led to a quite fast lookup and searching.
- The ability to handle very large volumes of data (e.g., greater than 100 MB or 1 Million entries per DSA) is not a requirement.
- Substantial data robustness is not required: there is no need to employ complex data backup techniques.
- Use (as opposed to provision) of Authentication services.

2.5 Roadmap

This manual is split into 6 parts. You are reading Part I, which is a general introduction. Part II describes a set of user interfaces (DUAs) developed as part of QUIPU. Part III, an administrators guide, describes how to set up both the DUAs introduced in Part II, and how the install and manage a QUIPU Directory System Agent (DSA). Part IV is a programmers guide which discusses a procedural interface to the directory for those of you who want to write your own DUAs. Part V is a discussion of some of the design issues not already covered elsewhere; this is essentially included for those of you who are interested in the DSA implementation. Finally, Part VI contains Appendices.

2.6 QUIPU Support Address

If you have any problem installing QUIPU, following the documentation or any other QUIPU related problems, then there are two discussion lists.

Comments concerning the operation of QUIPU should be addressed to the QUIPU support address:

Internet Mailbox:	`quipu-support@cs.ucl.ac.uk`
Janet Mailbox:	`quipu-support@uk.ac.ucl.cs`
X.400 Mailbox:	`surname = quipu-support`
	`ou = cs`
	`Org = UCL`
	`PRMD = UK.AC`
	`ADMD = Gold 400`
	`C = GB`

Or, you could look up the mailbox attribute of

```
c=GB
o=University College London
ou=Computer Science
cn=QUIPU-Support
```

in the Directory!!!

There is also a discussion list for a general discussion of topics related to QUIPU; the address is as above, but with "quipu-support" replaced by just "quipu" (e.g., quipu@cs.ucl.ac.uk). We suggest that everybody who is intending to run QUIPU should be on this list, as this will be used to keep you informed of what is happening. Details of updates will also be sent to this list.

If you would like to be added to the **quipu** discussion list, send a message to "quipu-request" (e.g., quipu-request@cs.ucl.ac.uk).

2.7 Acknowledgements

QUIPU was developed at the Department of Computer Science at University College London, under the ægis of the INCA (Integrated Network Communication Architecture) project, which is project 395 of ESPRIT (European Strategic Programme for Research into Information Technology). The partners of INCA (GEC plc, Olivetti, Nixdorf AG, and Modcomp GmbH) are acknowledged for releasing this software into the public domain.

Continued funding of QUIPU as Openly Available Software is provided by the Joint Network Team (JNT).

QUIPU 6.0 was implemented primarily by Colin Robbins and Alan Turland, with considerable help from Marshall Rose of Performance Systems International. Mike Roe, implemented the authentication code, and the T.61 string handling code.

After QUIPU 6.0, the core development has been from Colin Robbins. Tim Howes of University of Michagan has kindly donated the very valuable "TURBO" options to QUIPU. Alan Turland has added the asynchronous DUA interface.

Chris Moore of The Wollongong Group helped considerably in the early development of QUIPU, and integration with ISODE. Simon Walton of University College London, also provided much help in integrating the software with ISODE.

Steve Titcombe, of University College London, did much of the early work on DISH, and is now working on the management tools.

Paul Sharpe, of GEC Hirst Research Laboratories put considerable effort into the early development of SD. This work was continued at Brunel University by Andrew Findlay, Damanjit Mahl and Stefan Nahajski who have also developed the POD and XD interfaces.

Paul Barker of University College London, has designed and developed the "dsc" DUA interface, the DSA log processing scripts and has provided very valuable feedback throughout the project.

Whilst at UCL on secondment from CSIRO, Andrew Worsley put considerable effort into enhancing Pepsy to provide all the extra features the QUIPU needed to be able utilise the tool.

George Michaelson of the University of Queensland, Julian P. Onions of X-Tel Services Ltd, Andrew Findlay of Brunel University, Gier Pederson from the University of Oslo, Petri Jokela of Telecom in Finland, Juha Heinanen of the Tampere University of Technology, Peter Yee from NASA and Piete Brooks of Cambridge University, have all run various test versions of the system, and provided much useful feedback.

Kevin Jordan from The Control Data Corporation has given the G3 Fax handling code a much needed overhaul.

Part II

User's Guide

Chapter 3

The OSI Directory

This chapter is designed as a quick introduction to the model of X.500 directories, to provide the reader with sufficient information to be able to understand the following chapters.

3.1 The Model

The OSI Directory is intended to support human user querying, allowing users to find, *inter alia*, telephone and address information of organisations and other users.

It is also intended to support electronic communication such as message handling systems and file transfer. The Directory provides name to address mapping to support, for example, OSI presentation address look-ups. Message handling systems will be provided with support for user-friendly naming, security and distribution lists.

In essence the Directory is a database with certain key characteristics.

- The Directory is intended to be very large and highly distributed. It is anticipated that the Directory will be distributed largely on an organisational basis.

- The Directory is hierarchically structured, the entries being arranged in the form of a tree called the Directory Information Tree (DIT). An example DIT is shown in Figure 3.1. Entries near the root of the tree will usually represent objects such as countries (e.g., "GB") and

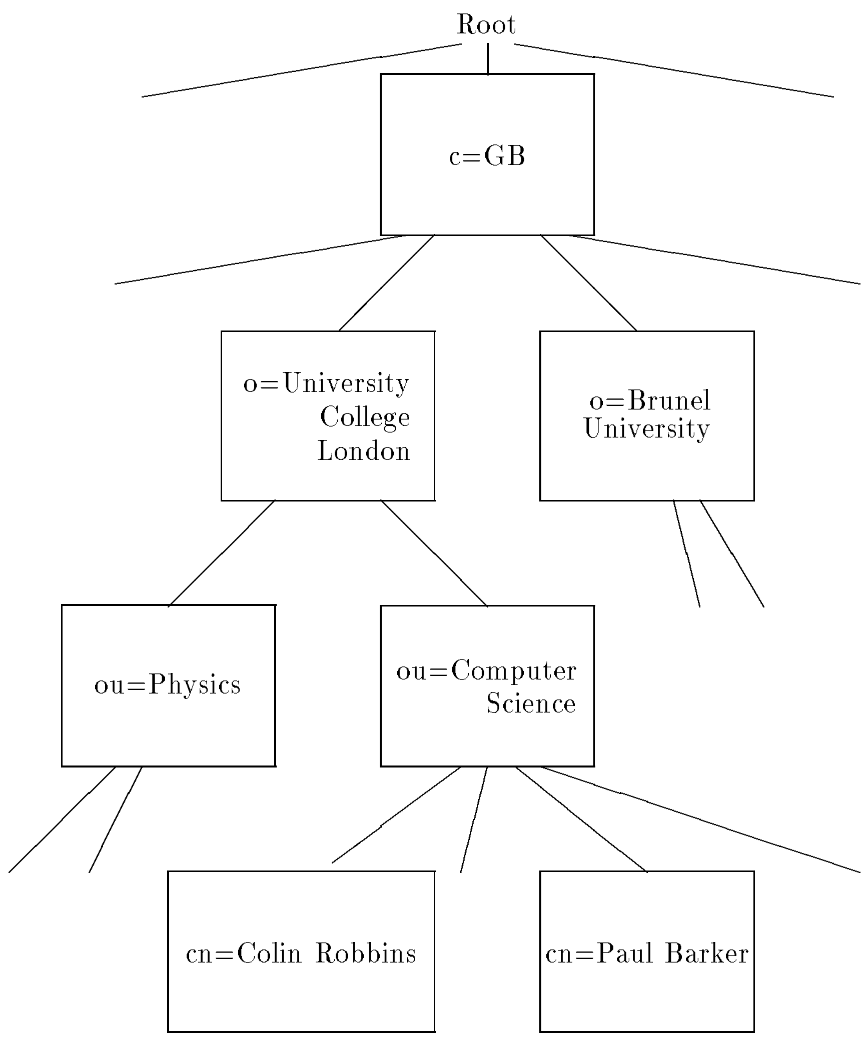

Figure 3.1: Example DIT

organisations (e.g., “University College London”), entries at or near the leaves of the tree will represent people (e.g., “Colin Robbins”), equipment or application processes.

- Read and search operations will dominate over modification operations.
- Temporary inconsistencies in the data are acceptable. This greatly facilitates the replication of data in the Directory by obviating concerns about record locking and atomic operations.

3.2 Information Representation

There are various structures needed to represent the data required in the directory database. These are described briefly below.

3.2.1 Object Identifiers

An important part of the OSI world is the Object Identifier (OID).

An OID is a hierarchy of numbers used to uniquely describe various objects within the OSI world: for example, “1.0.8571.1.1” describes the FTAM protocol; “0.9.2342.19200300.99.1” defines a QUIPU Attribute Type. The strings of numbers look “horrible” and can be difficult to remember. As such, QUIPU provides a mapping from OIDs to “strings”, so that easy to remember strings can be used in place of the numeric OIDs, for example “iso ftam” or “quipuAttributeType”.

The mapping are defined in a set of oidtables which are discussed in Section 14.11.

3.2.2 Attributes

The directory holds information about various entities, such as a person or an organisation. The information is held in an information object, typically referred to as an *entry*. An entry consists of a set of one of more attributes (see Figure 3.2).

An attribute is represented by an attribute type, and set of attribute values. An attribute with a single value is represented as follows:-

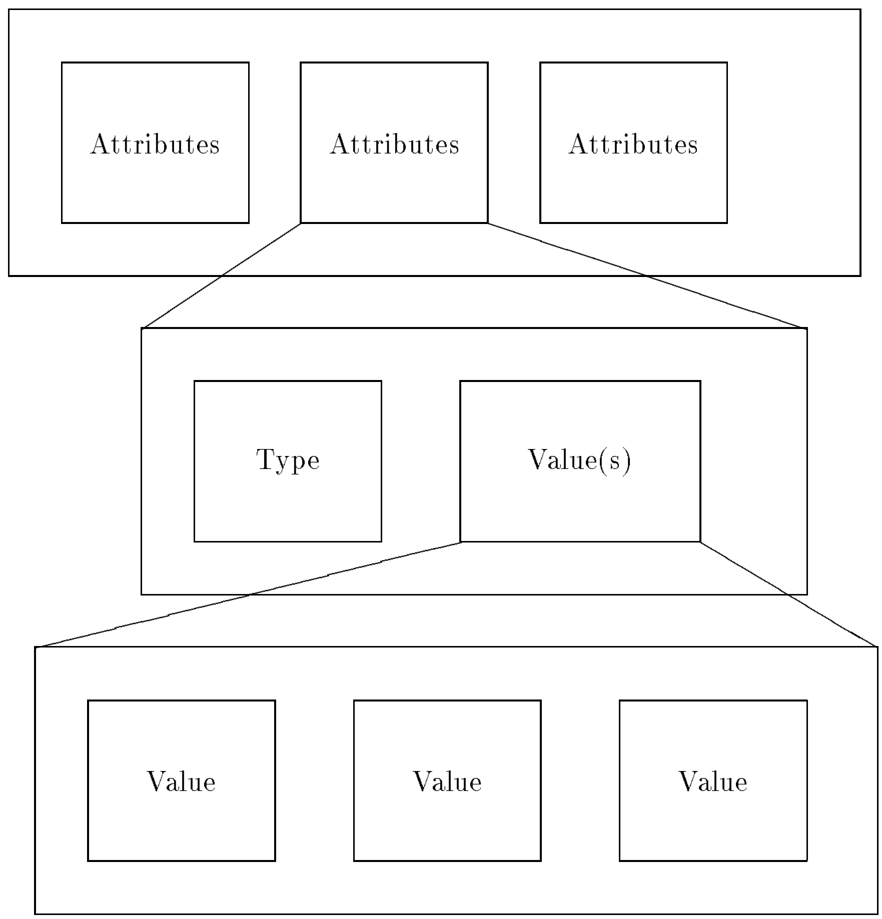

Figure 3.2: Structure of an Entry

<Attribute Type> "=" <Attribute Value>

The attribute type can either be a string (which must be in the attribute oidtable — see Section 14.11) or the OID in a dotted decimal format.

Each attribute value is represented by a string, the format of the attribute value depends upon the syntax defined in the oidtables, but in general will be a string.

So

```
roomNumber = G24
2.5.4.20   = 453-5674
commonName = Colin Robbins & Colin John Robbins
photo      = {ASN} 0308207b4001488001fd...
```

is an example of an entry, with 4 attributes. The "commonName" attribute is multi-valued, with the two values separated by the "&" symbol. Notice how the "photo" attribute does not have a specific string format, so a hexadecimal "ASN.1" representation is used. "2.5.4.20" is an Object Identifier represented in numeric form.

3.2.3 Names

Within an entry, some attributes (generally one) have special importance, and are called *distinguished attributes*. The collection of these attributes form the *Relative Distinguished Name* (RDN).

An RDN is usually represented by a single valued attribute, thus

```
commonName = Alan Turland
organization = University College London
```

are valid RDNs.

An RDN made up of multiple distinguished attributes uses the `%` symbol to separate the values, for example

```
userid = quipu % commonName = Colin Robbins
```

A *Distinguished Name* (DN) is the sequence of RDNs that uniquely define a node in the Directory Information Tree (DIT). These are represented as follows:-

RDN ["@" RDN ...]

So

```
countryName = GB @ organization = University College London
```

is an example of a valid DN.

In some cases an RDN or DN is specified relative to a node other than the root in the DIT. To be unambiguous a DN may be rooted using a leading "@", for example

```
@ countryName = GB @ organization = University College London
```

3.3 Directory User Agent

The Directory User Agent (DUA) is the entity you as a user will connect to when you interact with the Directory Service. It will help you formulate your queries, wrap them up in the required protocol, pass them to the Directory, and then show you the results obtained.

The X.500 standard defines only the protocol the DUA should use when taking to the Directory, and as such DUAs come in many varieties, but the basic concepts are the same.

A DUA talks to the directory using the Directory Access Protocol (DAP).

The following chapters of this part of the manual describe several such DUA interfaces available in the ISODE. If is left to system managers to decide which form of interface is appropriate at your site, and which to install. Below is a brief description of each interface.

dish: The DIrectory SHell
: This interface give a user full access to the DAP, and as such may be complex for novice users.

sid: Steve's Interface to Dish
: A set of scripts that make Dish usable for novice users.

dsc: Directory SCript
: A shell script using DISH aimed at novice users.

fred: FRont End to Dish
Fred provides a White Pages User Interface, which hides most of the complexity of the OSI directory.

pod: Pop Up Directory
A DUA for the X Window System[1].

xd: X Directory
Another DUA for the X Window System.

sd: Screen Directory
A Curses based DUA.

ufn: User Friendly Name
A very simple example interface that takes a UFN and returns a distinguished name [SKill90].

3.4 Directory System Agent

The Directory System Agent (DSA) is the entity that generates the results to requests from a DUA. It may handle the request itself, ask another DSA (using the Directory System Protocol), or advise the DUA to contact another DSA (see Figure 3.3). This part of the directory is discussed more in Parts III and V of this manual. Here we shall concentrate on the DUA.

[1]The X Window System is a trademark of the Massachusettes Institute of Technology

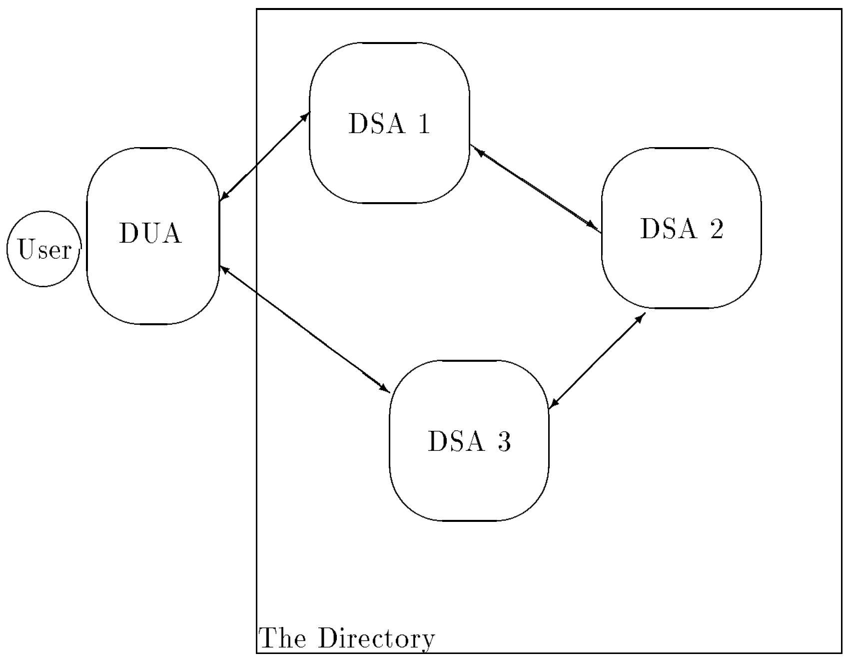

Figure 3.3: DUA/DSA Interaction

Chapter 4

DISH

This chapter describes DISH, a DIrectory SHell interface to the Directory. It provides an interface to the Directory in an similar manner to the way that MH provides an interface to a message handling systems. As with MH, dish can either be invoked as a single process with the full repertoire of commands built in, or it can be invoked by individual shell commands. This latter style allows dish to be used with others tools to provide very flexible access to the directory.

Dish provides a very powerful interface onto the Directory and gives a user access to the full Directory Access Protocol(DAP). The price of such comprehensiveness is complexity. Dish has a large number of flags, and both the syntax and volume of typing required can seem forbidding to the novice user. One compromise solution is to use dish to build interfaces which are easier to use and more intuitive, for example SID and DSC (see Chapters 5 and 6).

To run DISH in interactive mode invoke *dish*(1c), then issue any of the commands described in Section 4.1. This mode of operation is especially useful for novice users. The process of connecting to the Directory — binding — takes place automatically when the program is invoked, and an unbind is issued when you quit.

The arguments to *dish*(1c) are the same as for bind (see Section 4.1.13), with the addition of a "-pipe" flag which is used to start dish in "shell" mode (see Section 4.7) (NOTE: "dish -pipe &" has the same effect as the shell command "bind").

4.1 Commands

Each of the DISH commands described in this section has a large number of valid flags, the full names of the flags are given below, however the shortest unique name is sufficient to select the flag. Similarly, when dish is used interactively the shortest unique name is taken for the command name (e.g. "l" for "list"). There is an exception in that "sh" is interpreted as "showentry", "shown" must be typed for "showname".

The additional flag `-help` can be specified with every command to get limited runtime help.

4.1.1 Moveto

With nearly every command it is possible to supply the distinguished name of the object you want to reference. In the syntax used to describe the dish commands this is represented by

`<object>`.

An initial '`@`' in an object description specifies that the name is relative to the root of the Directory Information Tree (DIT), otherwise the name is taken as being relative to the current position in the DIT. The special name component ".." is used to mean "one position up" from the current position. Some examples of valid names and their interpretation relative to how a distinguished name is built up are shown below by the following sequence of names:

`"@c=GB@o=University College London@ou=Computer Science"`:
The specified object is
"c= GB @ o= University College London @ ou= Computer Science".

`"cn=Colin Robbins"`: describes the object cn=Colin Robbins relative to the current position ("c= GB @ o= University College London @ ou= Computer Science @ cn= Colin Robbins"). Spaces in a name will be seen as the start of a separate argument by the shell, so the name must be quoted.

`"..@cn=Steve Titcombe"`: describes the entry cn=Steve Titcombe at the same level in the DIT as the current position ("c= GB @ o= University College London @ ou= Computer Science @ cn= Steve Titcombe").

Objects can also be expressed in the form of sequence numbers and nicknames. See Sections 4.2 and 4.4.1.

When you specify an object, the current position in the DIT is not changed. To change the current position you should use the command

```
moveto [-[no]pwd] [-[no]check] <position>
```

The `-pwd` flag tells moveto to print the current position in the DIT. The `-nocheck` flag tells moveto NOT to check the entry exists, normally, moveto will invoke a '`read`' to check the named entry exists.

The current position can also be changed with the `-move` flag to showentry and list commands (see Sections 4.1.3 and 4.1.2).

4.1.2 Showentry

```
showentry [<object>] [-[no]cache] [-[no]name]
          [-[no]move]
          [<any of the read arguments>]
          [<any of the fred arguments>]
```

Showentry will display some or all of the attributes of the specified entry. A combination of the read argument flags and your */.quipurc* file described later in the chapter are used to define which attributes to show.

The `-noname` option tells showentry to show the distinguished name of the current entry as well as the attributes.

`-cache` is used the tell showentry to use cached information only — do not issue a read operation. This is used to see what the DUA has cached, and then get information when the local DSA is unavailable.

`-nocache` is used to enforce a directory read, this is used to update the cache directly.

`-move` is used to tell dish to change the current position in the DIT to the object specified as well as showing the requested attributes.

See Section 4.1.4 for a discussion of the flags for fred.

read flags

These flags are used by showentry(1), showname(1) and search(1).

```
[-[no]types <attribute-type> *] [-[no]all]
[-[no]value] [-[no]show]
[-[no]key] [-edb]
[-proc <syntax> <process>]
```

The `all` flag request that all attribute are read (default).

The `value` flag reads the attribute value, which is the default, the inverse is to read the attribute types only.

The `novalue` flag says print the attribute types only.

The `show` flag is used to make dish show the requested attributes (this is the default for read, but not search).

The `key` flag determines whether the key (attribute type) will be shown along with the attribute value.

The `edb` flag requests that the data is displayed in "EDB" format, that is, a format which can be used in QUIPU data files. This is not such a nice format to present users, but is more easily managed in shell scripts.

The `type` flag requests that only the supplied attributes are read from the DSA, the inverse of this `-notype` does not prevent the attributes being read (as there is no easy mechanism to perform this within X.500), but it does stop them being shown.

The `proc` flag is used to instruct the display routines to uses an external process to display the attribute. For example if you have a process that displays presentation elements representing "OID's" in a particular way, then you can use

```
showentry -proc OID /usr/bin/oid_display_process
```

then all OIDs will be displayed using "/usr/bin/oid_display_process". The process should read the presentation element from the standard input, and place any textual results on the standard output.

4.1.3 List

```
list [<object>] [-nocache] [-noshow] [-[no]move]
```

This function displays the relative distinguished names of the children below your current position in the DIT. Typically, if there are a large number of children, you will not be able to list them all, as a "sizelimit" will prevent you. For example

```
Dish -> list
1   commonName=Camayoc
2   commonName=Colin Robbins
3   commonName=Jess
4   commonName=Julian Onions
5   commonName=lancaster
6   commonName=PP Support
7   commonName=QUIPU Support
8   commonName=Tony Roadknight
9   commonName=Hugh Smith
10  commonName=Peter Cowen
(Limit problem)
Dish ->
```

shows a size limit of 10. Using the `-sizelimit` service control you can often increase this limit upto some administrative limit (typically 100) set by the DSA you are connected to.

`-nocache` tells list to re-read the directory — do not use cached information.

`-noshow` tells list not to display the names found, but still construct sequences (see Section 4.2).

`-move` is used to tell dish to change the current position in the DIT to the object specified.

4.1.4 Search

```
search  [[-object] <object>]
        [-baseobject] [-singlelevel] [-subtree]
        [[-filter] <filter>]
        [-[no]relative] [-[no]searchaliases]
        [-[no]partial] [-hitone]
        [<any of the read arguments>]
        [-fred [-expand] [-full] [-summary]
               [-nofredseq] [-subdisplay]]
```

Search the DIT starting at the object specified, for entries that match the given filter. When an entry is found to match the filter, only distinguished name is printed unless a '`-show`' option is specified, when the attributes are displayed as well.

If no filter is specified a default filter (which matches any object in the DIT) is used. This operation has a similar effect to a list operation, but attributes can be shown at the same time.

If no flags are given to a search command, then only one parameter is allowed. This is taken to be the filter and NOT the base object as in ALL the other commands. The flags `-filter` and `-object` are supplied to allow the user to specify if both a filter and base object (NOTE only one need be flagged). For example, the following are all valid search commands

```
search
search <filter>
search -filter <filter>
search -object <object>
search -filter <filter> -object <object>
search <filter> -object <object>
search -filter <filter> <object>
```

Whereas

```
search <object>             (interpreted as a filter !)
search <filter> <object>
```

are not valid.

The three flags '`-baseobject`' '`-singlelevel`' and '`-subtree`' are mutually exclusive. They are used to control the depth of the search operation. The '`-baseobject`' option searches only the specified object. The '`-singlelevel`' option specifies all siblings of the specified object should be searched. The flag '`-subtree`' searches all the levels below and including the current position recursively. The default is '`-singlelevel`'.

For example:-

```
search -object
   "@c=GB@o=University College London@ou=Computer Science"
   -subtree -filter "cn=colin robbins" -type telephoneNumber
   -show
```

will search the DIT subtree below "c=GB @ o=University College London @ ou=Computer Science" for an exact match of "colin robbins", and then call showentry to display the *telephoneNumber* attribute.

Class	Characters
1	BFPV
2	SCGJKQXZ
3	DT
4	L
5	MN
6	R
0	all others (ignored)

Figure 4.1: Soundex Character Classes

The `-norelative` flag is used to tell `showname` to print the full distinguished name of the results, otherwise the name relative to the current position is printed.

The `-[no]searchaliases` flag directs the search as to whether aliases encountered in the search should be dereferenced and thus searched. This is different to the `-[dont]dereferencealias` service control which defines what happens to aliases found in locating the base object of the search, whilst the `-[no]searchaliases` controls aliases encountered in the siblings of base search object. The default is "`-dereferencealias -nosearchaliases`".

The `-hitone` flag is used to tell Dish to consider it an error if the search returns more than one entry. This is particularly useful when using Dish from shell scripts (see Section 4.7).

From time to time, search will not be able to do the entire search, and will return partial results, the `-nopartial` directs dish not to print the partial result information. To see the full set of partial results, both `-partial` and `-show` are needed, otherwise only a summary is printed. Unlike referral errors, Dish does not follow these partials references.

More on Matching

When searching for an exact match ("=") the "`*`" character is used as a wildcard, and a substring match takes place (unless only a "`*`" is specified, when a *presence* match is used —- hence any entry containing that attribute will be matched). For example, "`Colin*`" will match any attribute value starting with the string "`Colin`" and "`*Robbins`" any attribute value ending in the string "`Robbins`". When constructing substring filter, is should be noted that a QUIPU DSA can resolve searches of the form "`string*`" significantly faster than searches of the "`*string*`" or "`*string`" form.

Remember a large number of the standard attributes match case independently, for example "x-tel" will match "X-Tel".

As well as an exact equality match, the following types of match can be specified

```
~=      Approximately equals.
>=      Greater than or equal.
<=      Less than or equal.
```

The Approximate match method used is DSA dependent. A QUIPU DSA uses a soundex based algorithm. This works by grouping similar sounding characters into classes. The classes and their corresponding characters are as shown in Figure 4.1. The first character of a word is always used as the first character of its corresponding soundex code. Adjacent similar characters are ignored. Thus, the word "Robbins" has soundex code "R152". Since the word "Robens" also has soundex code "R152", these two words are approximately equal. To match multiple words each of the target words must appear in order in the string to which it is being compared. There may, however, be other things in between the words matched. For example: "Tim Howes" would match "Timothy Alan Howes" since "Tim" matches "Timothy", "Howes" matches "Howes", and the matched words are in the proper order.

Filter items can be linked with *and* "`&`", or *or* "|". Brackets "`( )`" should be used to enforce the boolean ordering of the expressions, otherwise the evaluation is left to right. A *not* "`!`" can be used to specify the boolean not operator.

If a filter does not have a type specified e.g., "-filter steve", then an approximate match for the specified common name is assumed (in this case "-filter cn~=steve" is assumed).

A more complex example of a filter that searches for anybody whose is a member of staff or a student, who has a "drink" attribute, whose name approximately matches "steve", but whose surname is not "Kille":-

```
cn~=steve & (userClass = staff | userClass = student) &
            (! surname = Kille) & drink = *
```

A list of all the attributes that can be used to search for an entry is given in Chapter 10, together with a description of the type matching of matching allowed.

fred flags

These flags are used by fred, and are activated when the `-fred` flag is given.

The `-expand` flag indicates that the any matched entry should be displayed in full followed by children entries.

The `-full` flag indicates that if more than one entry is matched on a search, then all entries should be displayed in full.

The `-summary` flag indicates that a one-line display should be used for the entry.

The `-subdisplay` flag indicates that a one-line display should be used for the entry, followed by the entry's children.

The `-nofredseq` flag indicates that the entry (and children) should not be added to the current sequence.

The `-fredlist` flag indicates that any attributes containing DNs should be flagged when using the DA-service.

4.1.5 Add Entry

```
add [<object>] [-draft <draft> [-noedit]] [-template <draft>]
    [-newdraft] [-objectclass <objectclass>]
```

Add is used to add entries to the DIT. This invokes editentry on the draft entry. If there is not a draft entry specified, a draft entry of the specified objectclass (default — thornperson & quipuObject) is created. The default draft file is ".dishdraft" in your home directory, this can be altered with the `-draft` option.

When editing has finished an add operation is sent to the directory (the `-noedit` flag following a `-draft` stops the editor being invoked).

`-newdraft` causes the current draft to be overwritten with a new template of the appropriate objectclass.

`-template` is used to specify a template that should be used during editing.

On successful completion of the add, the draft entry is renamed with a suffix of ".old".

4.1.6 Editentry

```
editentry <draft>
```

This option is only available from the shell, and not from the Dish program. It is generally not invoked directly by a user, but by the other dish commands.

editentry invokes an editor (as defined by the users $EDITOR environment variable) on the current draft entry. In a future release it is hoped to have an editor that knows about the syntax of an entry, and thus ensures that the entry you have supplied is syntactically correct.

4.1.7 Delete Entry

```
delete [<object>]
```

The specified entry is removed from the DIT (remember that X.500 only allows leaf entries to be deleted).

4.1.8 Modify Entry

```
modify [<object>] [-draft <draft> [-noedit]] [-newdraft]
       [-add <attribute type>=<attribute value>]
       [-remove <attribute type>=<attribute value>]
       [<any of the showentry arguments>]
```

Modify is used to modify existing entries in the DIT, and has two modes of operation. The first is used to modify specific attributes and uses the `-add` and `-remove` flags to add and remove single attributes and values. Many of these can be strung together in the same command, e.g.,

```
modify -add "description=new attribute" -remove "drink=Chocolate"
```

The second method is used for altering many attributes at the same time. Initially "`showentry -all -edb -nocache`" is used to get the current DIT entry, and place a copy of if in your .dishdraft file. If the `-draft` option is specified then the given file is used. After editing, any changes to the entry are sent to the directory (the `-noedit` flag following a `-draft` stops the editor being invoked).

The draft file is handled in the same way as with add, except, when a new draft is created, the current values of the attributes are read from the directory.

4.1.9 ModifyRDN

```
modifyrdn [<object>] -name <newrdn> [-[no]delete]
```

This is used to modify the RDN of an entry. The RDN can not be changed using the modify operation (remember, as with delete X.500 only allows this operation on leaf nodes).

The `-nodelete` flag is used to prevent the old RDN being removed as an attribute of the entry.

4.1.10 Showname

```
showname [<object>] [-[no]compact] [-[no]cache] [-[no]ufn]
         [<any of the read arguments>]
```

A name can be printed either in compact form (as seen in EDB or dish-draft files), just showing the distinguished name, in a UFN format (default) or by showing the distinguished attributes one per line.

4.1.11 Compare

```
compare [<object>] -attribute <attribute> [-[no]print]
```

For example

```
compare userclass=student
```

This command has a return value of true or false (1 or 0).

If '`-print`' is specified the strings "TRUE" or "FALSE" are printed.

4.1.12 Squid

```
squid [-sequence [<name>]]
      [-alias <object>] [-version]
      [-user] [-syntax]
      [-fred]
```

The command *squid* (Status QUIpu Dish) with no parameters will inform you of the current status of your dish process for example:

```
Connected to Armadillo at Internet=128.40.16.220+2005
Current position: @c=GB@o=Nottingham University
User name: @c=GB@o=University College London@
           ou=Computer Science@cn=Colin Robbins
Current sequence 'default'
```

`-sequence` makes squid print the specified sequence. Sequences are described in Section 4.2.

`-alias` adds the given DN to the current sequence.

`-version` print the version number of the dish process and associated ISODE and DSAP libraries.

`-user` print the DN you are bound as.

`-syntax` print the Directory syntaxes known to dish.

`-fred` must be the first argument if supplied, it causes any Distinguished names printed by the command to be shown using the user-friendly naming notation rather than the DN notation.

4.1.13 Bind

```
bind [-noconnect] [[-user] [<username>]]
     [-password [<password>]] [-[no]refer]
     [-call <dsaname>]
     [-simple] [-protected] [-strong] [-noauthentication]
```

This is used to (re)connect to the directory. This is not normally required, as each command will bind as necessary, but allows advanced users to contact specified DSAs and change their username.

`-noconnect` is used to start dish (and cache) without connection to a directory.

`-[no]refer` is used to tell Dish whether to automatically follow referrals issued by the DSA, by default they are followed.

To connect to the directory you need to be authenticated. The are four flags to control the level of authentication dish attempts to use. The lowest level of authentication is "anonymous", which is used when you do not supply a Distinguished Name. A DSA may respond with an "inappropriate authentication" which means you should use a higher level of authentication. The next level of authentication is achieved by supplying a Distinguished Name only, with no password. The `-noauthentication` is use to tell dish not to prompt for a password. The next levels are simple and protected simple authentication which use the `-simple` and `-protected` flags respectively. These present a textual password to the directory which the directory the uses to authenticate you. Using `-simple` the password is sent across the network "in the clear", whereas `-protected` encrypts it, thus you are advised to used `-protected` if your directory supports it.

The `-strong` flag tells Dish to send strong authentication credentials to the directory. This is not fully implemented in this version of QUIPU.

To connect to the DSA using "simple" or "protected" authentication, you must supply a distinguished name (`-user`) and password (`-password`). This however reveals your password to anybody who is looking at your terminal, but is useful for issuing bind operations from a shell script. To prevent this you need to place the password in your UNIX protected ".quipurc" (see Section 4.4.1). Alternatively, if you do not supply a password, you will be prompted for one (with echoing turned off).

4.1.14 Unbind

`unbind [-noquit]`

This is used to break the connection to a DSA. If `-noquit` is specified, the dish process does not die (This is used to maintain the cache).

To be "friendly" to other users, you should unbind when you have finished using the directory — put an `unbind` in you ".logout" file if you use `dish` from the shell.

4.1.15 Fred

```
fred [-display <name>]
     [-dm2dn [-list] [-phone] [-photo] <domain-or-mailbox>]
     [-expand [-full] <DN>]
     [-ufn [-list,][-mailbox,][-phone,][-photo,][-options %x,]<name...>]
     [-ufnrc <list...>]
```

This is used by the fred program to provide some functions which would require complicated interactions between fred and dish.

`-display` is used to set the address of an X window display.

`-dm2dn` is used to map a domain name from the DNS or a mailbox into a distinguished name in the Directory. The `-list` suboption returns a list of possible matches, `-phone` returns the phone number attribute of the matched entry, `-photo` returns the photo attribute of the matched entry.

`-expand` is used to see if a node has children.

`-ufn` is used to perform a user-friendly naming match. The `-list,` sub-option returns a list of possible matches, `-mailbox,` returns the mailbox attribute of the matched entry, `-phone,` returns the phone number attribute of the matched entry, `-photo` returns the photo attribute of the matched entry, `-options` is used to customize the behavior of the matching algorithm.

`-ufnrc` is used to set the search-list for user-friendly naming.

4.2 Sequences

Results returned by *search* and *list* operations may be long, and for this reason have a reference number printed beside them. The reference number can be used as the "object" in any of the calls to the directory. Thus

```
showentry 6
```

shows the entry labelled with the sequence number "6" by a previous *search* or *list* operation.

When you come into dish by default, the resultant numbers of list and search operations are added to a sequence called "default".

Two flags

```
[-sequence <name>] [-nosequence]
```

are used to control sequences. The "`-nosequence`" flag is use to completely remove the concept of sequences (so no numbers are printed by list or search).

To get a different sequence (hence renumber from 1 again), you can change this to "mysequence" with a

```
-sequence mysequence
```

flag.

The special keyword "`reset`" is used to reset the current sequence, thus causing future operations to be renumbered from 1. This is different to setting a new sequence, as the old sequence values are removed.

Occasionally you will want to search an entry defined in one sequence, putting the results in another, for this purpose the special token "result" is recognised, thus

```
search 4 -sequence xxx -sequence result yyy
```

will search entry 4 as defined by sequence "xxx", but place the results in sequence "yyy".

4.3 Service Controls

All the commands described in Sections 4.1.2 through 4.1.11 have additional flags to control the type of service provided. The advice to a novice user is let these flags take their default values.

The flags recognised are listed below:-

`-(no)preferchain`: Advise the DSA (not) to chain the operation if required — The DSA **IS** allowed to ignore the advice!

`-(no)chaining`: (Prohibit) Allow the use of chaining.

`-(dont)usecopy`: (Prohibit) Allow the use of a cached or slave copy of the data. Note this refers to data cached in the DSA, there are separate flags (already described) for data cached in the DUA.

`-(dont)dereferencealias`: (Do not) dereference aliases if found in the path of a query.

`-low`: Flag the query as low priority.

`-medium`: Flag the query as medium (default) priority.

`-high`: Flag the query as high priority.

`-timelimit` n: Set the time limit to n seconds.

`-notimelimit`: Do not specify a time limit.

`-sizelimit` n: Set the size limit to n entries.

`-nosizelimit`: Do not specify a size limit.

`-(no)localscope`: (Do not) limit operation to local scope.

`-(no)refer`: (Do not) automatically follow referrals issued by the DSA.

`-strong`: Ask the DSA to use strong authentication when sending the results.

4.4 Tailoring

There are two levels of tailoring that control the operation of the DISH DUA. First of all there is the system wide tailor file *dsaptailor* (found in the ISODE `ETCDIR` directory, usually */usr/etc/*), and then users private tailor file *.quipurc*.

The file *dsaptailor* is described in Section 13.2.

4.4.1 .quipurc

The file *.quipurc* is read by `dish` on start up, and has the following format:-

```
flag: value
```

The following flags are currently recognised:-

username: The name of the user to bind as.

password: The password to use when binding. For this reason care should be taken to ensure you .quipurc file is not publicly readable.

cache_time: How long to keep the dish process alive after unbinding (specified in minutes)

connect_time: How long to keep a connection open, it there is no activity (specified in minutes)

service: A list of default service control flags (as defined in Section 4.3).

sequence: Add the supplied DN parameter to the default sequence.

dsap: This can be followed by one of the dsaptailor options described in the Section 13.2.

isode: This can be followed by one of the ISODE tailor options described in *Volume One* of this manual.

notype: The argument is expected to be an attribute type. This attribute will not be displayed by showentry by default.

type: The only valid argument for this entry is “unknown”, and says that unknown attributes should be shown, by default, they are not shown.

certificate: Used for strong authentication.

secret_key: Used for strong authentication.

<**dish command**>: A list of default flags to be used when “command” is used. For example

```
showname: -compact
```

says that “showname” should always be invoked with the `-compact` flag set.

<**nickname**>: A nickname entry, for example

```
cjr: "c=GB@o=X-Tel Services Ltd@cn=Colin Robbins"
```

This can then be used in distinguished name arguments, for example

```
showentry cjr
```

will show the entry for “Colin Robbins”.

4.5 Remote Management of the DSA

```
dsacontrol [-[un]lock <entry>] [-dump <directory>]
           [-tailor <string>] [-abort]
           [-restart] [-info] [-refresh <entry>]
           [-resync <entry>] -slave [<entry>]
```

If the *dish* program is bound to the DSA as the Directory manager (See Section 4.1.13 on binding), then it may be used to (primitively) manage the DSA. This feature is discussed in [SKill89b] and Section 14.3.1 of this manual.

`(un)lock` used to (un)lock subtrees of the DIT held locally. This (allows) prevents users to modify the DIT.

`dump` causes a copy of the local DIT to be dumped into the specified directory. Note that when the DSA dumps its in-core database, it does so relative to the (possibly remote) UNIX directory from which the DSA was started, not the directory that *dish*(1c) was invoked in! So it is best to specify path names in full to ensure the database is dumped where you expect it to be!

`tailor` is used to pass tailor command to a running DSA. The format of "string" is the same as a line in the "quiputailor" file. For example:-

```
dsacontrol -tailor "dsaplog level=all"
```

will turn the DSA logging on full.

`refresh` tells the DSA to re-read sub-tree "entry" from disk. As with all "dsacontrol" commands, "entry" must be the full DN of the target entry, the dish concept of "current position" is not used. Also, if relating to a disk operation, the CASE of the DN must be correct as well.

`resync` tells the DSA to re-write the sub-tree "entry" to disk.

`slave` tell the DSA to update its SLAVE EDB files. The keyword "root" is used to identify the root EDB file. The keyword "shadow" is used to tell the DSA to update any entries it has shadow copies of. See Section 14.10 for details.

`abort` tells the DSA to stop. `restart` tells the DSA to stop, then restart.

`info` asks the DSA to return some statistics about its database.

The error messages returned from a DSA are somewhat obscure when using dsacontrol. There are four things dish may report:

`DONE:` the operation has been successfully completed;

`Scheduled:` the operation has been scheduled, and will be completed as soon as possible;

`Access rights insufficient:` you are not bound as the DSA manager;

`Unwilling to perform:` the DSA was unable to perform the task, this generally means the argument to dsacontrol was wrong.

4.6 Caching in the DUA

All results returned from read and list operations are cached in memory in the DUAs. The cache is then used if possible to answer queries. The cache is kept for "cache_time" minutes as specified in the tailor files.

Disk caching has not been implemented yet, but it is intended to extend the in-core caching, so that the cached data can be written to disk. When the DUAs start up, the user will have the option to read this cache. Similar support for private data may be provided.

4.7 Running DISH from the Shell

Each of the dish commands mentioned in this chapter to 4.5 can be applied directly to the shell, without having to first invoke *dish*. This has the advantage that the result can be piped through programs like *more* thus giving the user flexibility in how they view the data.

Chapter 12 describes how to install this "extra" feature of dish.

Each time a new shell is invoked and a dish command entered, a new DISH will be invoked. This is sometimes not required. To prevent this from happening an environment variable "DISHPROC" can be set, and should be the TCP address to use for communication, for example

```
127.0.0.1 12345
```

where 127.0.0.1 is the IP address, and 12345 is the TCP port. The two example scripts show how the TCP port may be generated if required under `sh`. For `csh` you might use

```
if (${?DISHPROC} == 0) then
        setenv DISHPROC "127.0.0.1 `expr $$ + 10000`"
endif
```

4.7.1 Dishinit

A program called *dishinit* can be use to bind to the directory as the manager, read certain information and create a default `.quipurc` file for a specified user. This is a useful utility for new databases. If the `dsaptailor` variable `dishinit` is set to `on`, dishinit will be invoked by `dish` whenever is can not find a .quipurc file.

`dishinit` needs three parameters to be set in order to bind as the manager, and search the local subtree. These parameters are defined at the top of the dishinit script and are:-

manager: The DN of the manager of the local DSA

password: The managers password (the script should be protected with UNIX file protection).

position: The DN of the local sub-set of the DIT. This is where the entries for users will be looked for

4.7.2 Example Scripts

Two example scripts have been supplied as part of ISODE, and can be found in the *others/quipu/uips/dish* directory.

dsaping: contacts all the DSAs at a given level in the DIT, and gathers some statistics.

dsalist: produces a list of all the (known) DSAs in the DIT.

4.7.3 Files

When the multiple command version of dish is compiled on a machine that does not support sockets, name pipes are used to communicate with the a background process.

The file "/tmp/dish<pid>" is used by dish to write results to the calling processes, where the "pid" is the process id of the calling process.

The file "/tmp/dish–<tty>–<uid>" is used by calling process to send commands to dish, where "tty" is the users terminal number, and "uid" their user id.

Chapter 5

SID

SID (Steve's Interface to Dish) is a Directory User Agent, designed to be incredibly easy to use. It is implemented as a simple set of scripts to the DISH DUA.

There are a lot of flags to DISH, this can make it awkward to use, and in general shows too much directory specific information. Fred has a centralised view of the world, and does not give the advantages of a shell based interface.

SID is a simple approach of giving appropriate aliases to access DISH commands. It is targetted for searching for people and organisations. It may be useful "as is" to others, and also give an indication as to how DISH may be used.

5.1 Quickstart

WARNING Before you start to try SID, you must have the variable DISH-PROC set in your environment. The mechanism suggested in Section 5.6 is a good way to achieve this. To experiment, csh users can type the incantation:

```
% setenv DISHPROC "127.0.0.1 `expr $$ + 1000`"
```

To find users within your Organisation is very easy. The default setup is optimised for your particular Organisational Unit. Simply use the command **psearch** with a single argument to identify the user looked for. This will identify all users with this key as a substring. For example:

```
% psearch steve
countryName=GB
organisation=University College London
department=Computer Science

1   cn=Steve Kille
telephoneNumber    - +44-1-380-7294 (work)
telephoneNumber    - 01-350-2888 (home)
2   cn=Stephen Titcombe
telephoneNumber    - +44-1-387-7050 x 3674/5
telephoneNumber    - Term = 01-388-4741
3   cn=Stephen Sackin
4   cn=Steve Wilbur
telephoneNumber    - +44-1-380-7287
5   cn=Stephen Usher
telephoneNumber    - +44-1-387-7050 x 3674/5
6   cn=Kevin Steptoe
7   cn=Tomasz Stepniak
8   cn=Stefan Penter
telephoneNumber    - +44-1-387-7050 x 3674/5
9   cn=Stephania Loizidou
10  cn=Steve Hodges
telephoneNumber    - +44-1-387-7050 x 3674/5
11  cn=Steve Davey
telephoneNumber    - +44-1-387-7050 x 7280 or 7289
12  cn=Steve Chan
telephoneNumber    - +44-1-387-7050 x 3674/5
13  cn=Steven Britton
telephoneNumber    - +44-1-387-7050 x 3715
14  cn=Stephen Beckles
telephoneNumber    - +44-1-387-7050 x 3674/5
15  cn=Steven Bacall
telephoneNumber    - +44-1-387-7050 x 3674/5
telephoneNumber    - dd 01-387-6978 (Campbell House)
```

Each match will have an associated number. To show a given entry, just use the command **showentry**, with the number identifying the entry as the single argument. For example to show entry 1 (Steve Kille):

```
% showentry 1
```

```
c=GB@o=University College London@ou=Computer Science@cn=Steve Kille
Address            - 35 Elspeth Road
                     London
                     SW11 1DW
userClass - csresstaff
photo - (See X window, pid 598)
roomNumber         - G24
favouriteDrink     - Pinta - Brakspears
info               - Directory worker
rfc822Mailbox      - S.Kille@cs.ucl.ac.uk
textEncodedORaddress - /Pn=Stephen.Kille/Ou=cs/O=ucl/ \
                     Prmd=uk.ac/admd=gold 400/c=gb/
userid             - steve
telephoneNumber    - +44-1-380-7294 (work)
telephoneNumber    - 01-350-2888 (home)
description        - New Description
surname            - Kille
name               - Steve E. Kille
name               - Stephen Kille
name               - Steve Kille
```

If you are running the X Window System[1], a picture will be displayed if it is present.

If you wish to search all of your Organisation, simply give a second argument to **psearch**. For example, to search UCL give "ucl" as shown:

```
% psearch baker ucl
Searching for People matching "baker" under:
countryName=GB
organisation=University College London

4   ou=French Lang and Lit@cn=F Baker
telephoneNumber  - 3077
5   ou=Economics@cn=V Bashkar
telephoneNumber  - 2286
6   ou=Biochemistry@cn=J Basra
telephoneNumber  - 2185
7   ou=Bartlett School@cn=H Bowker
telephoneNumber  - 7453
8   ou=Anatomy and Embryol@cn=D Becker
```

[1] The X Window System is a trademark of the Massachusettes Institute of Technology

```
telephoneNumber  - 3293
9   ou=Computer Science@cn=Malcolm Booker
telephoneNumber  - +44-1-387-7050 x 3645
10  ou=Computer Science@cn=Imtiaz Bashir
telephoneNumber  - +44-1-387-7050 x 7304
11  ou=Computer Science@cn=Benjamin Bacarisse
telephoneNumber  - +44-1-380-7212 (work)
telephoneNumber  - 01 987 2746 (home)
12  ou=Surgery@cn=L Baker
telephoneNumber  - 82-5255
13  ou=History Of Art@cn=B A Boucher
telephoneNumber  - 7210
14  ou=Geological Sciences@cn=J R Baker
telephoneNumber  - 2382
15  ou=Secretary's Office@cn=I H Baker
telephoneNumber  - 3000
```

5.2 Example Usage

A typical sequence to find someone outside your organisation is now shown:

```
% osearch nott
Searching in country "gb" for Organisations matching "nott"
1   o=Nottingham University
telephoneNumber  - +44-0602-484848x2862
2   o=NPL
3   o=JNT
telephoneNumber  - +44-0235-445724
% psearch smith 1
Searching for People matching "smith" under:
countryName=GB
organisation=Nottingham University

4   ou=Computer Science@cn=Hugh Smith
telephoneNumber  - extn 2862 (Departmental Secretary)
% showentry 4
c=GB@o=Nottingham University@ou=Computer Science@cn=Hugh Smith
photo            - (See X window, pid 1076)
```

```
telephoneNumber  - extn 2862 (Departmental Secretary)
description      - lecturer
surname          - Smith
name             - Hugh.Smith
name             - Hugh Smith
```

5.3 Use of Nicknames

It is convenient to have nicknames (private aliases) set up for commonly accessed parts of the Directory Information Tree. These can be set up in the QUIPU Profile (see Section 5.6), which sets up the nickname "us". Two nicknames used in the examples are:

ucl: Country GB; Organsisation University College London.

cs: Country GB; Organsisation University College London; Organisational Unit Computer Science.

Aliases may be used as an alternative to sequence numbers.

5.4 SID Commands

The initial SID commands are:

osearch: (Organisation Search) Search for an organisation. The first argument is the organisation being searched for. For example:

```
% osearch nott
```

This searches for any organisations approximating to this string, or containing it as a substring.

The second (optional) argument is the two letter country code (e.g., GB, US, ES, etc.). Use **clist** to find these codes. The default country is the previous one searched. Example:

```
% osearch psi us
```

psearch: (Person Search) Search for a person. The first argument is a key to identify the person. Approximate and substring matches are made. There are two ways of using psearch.

1. Search an identified organisation, using the second argument to osearch. Examples where the organisation is identified by a nickname:

   ```
   % psearch kirstein ucl
   ```

   Example where the organisation is identified by a sequence number, returned by a previous osearch command.

   ```
   % psearch steve 29
   ```

2. Move explicitly to an organistion, and then use psearch with only one argument. This is useful when repeated searches will be made from one point. Example:

   ```
   % moveto  ucl
   % psearch kirstein
   ```

clist: (Country List) Used to give a list of countries, with friendly descriptions, as well as the two letter code.

ousearch: (Organisational Unit Search) Search for an OU. Analogous to psearch. This is appropriate for use where organisations are large, and a full search takes too long.

dlist: (Directory List) This uses the search command to provide a listing of directory information, without a clutter of wild South American animals.

5.5 Standard DISH commands

The following DISH commands are useful "as is" for normal users, and are considered to be a part of SID.

showentry: Show an entry. The sinle argument is typically the numeric key returned by a search. This will be used a lot, and it is likely to be worthwhile setting up an alias. The author uses "ds" (directory show). Example, using a sequence number returned by a search.

```
% showentry 36
```

moveto: Move to a defined location. Example of moving to a nickname defined location:

```
% moveto ucl
```

modify: To modify your own entry, this command is used with the single argument "me". For example:

```
% modify me
```

5.6 QUIPU Profile

This is a suggested template for ".quipurc", note that this is very similar to the default file installed by *dishinit* described in Section 4.7.1.

```
username:c=GB@o=University College London@ou=Computer Science@cn=Steve Kille
me:c=GB@o=University College London@ou=Computer Science@cn=Steve Kille
password:steve

position: @c=GB@o=University College London@ou=Computer Science
notype: acl
notype: treestructure
notype: masterdsa
notype: slavedsa
notype: objectclass
notype: lastmodifiedby
notype: lastmodifiedtime
notype: userpassword
cache_time: 30
connect_time: 2

us: c=us
moveto: -pwd
showentry: -name
```

You must set the environment variable DISHPROC. This can be done by the following in the .login file (csh).

```
if (${?DISHPROC} == 0) then
        setenv DISHPROC "127.0.0.1 `expr $$ + 1000`"
endif
```

Chapter 6

DSC

dsc (which might stand for Directory SCript) is a Directory user interface originally written as UCL's public access user interface at University College London. It is primarily aimed at the novice user, although the user is allowed to select a more advanced interface, sd, if the user feels confident about using the Directory. The simple interface is restricted to looking up email addresses and telephone numbers for people. In order to keep the number of questions the user is asked to a minimum, the interface currently only searches within a single country. Clearly the interface could be tweaked to allow a wider scope of querying.

The interface is written as a *dish* script. It is regretted that the interface only runs on Bourne-style shells, which support functions. An enhanced version of dsc written in C is currently under development.

6.1 Starting up

On invoking the interface, a preamble is displayed showing the user the form of the questions which they will be asked to enter. The user is then asked to select a style of interface thus:

```
easy or advanced interface? (e/a):
```

where the default is "easy". See the Chapter refsd on *sd* for a description of the "advanced" interface.

6.2 A Simple Local Search

The user is now asked the following series of questions:

```
Enter the person's name (or "?" for help, "q" to quit):
Enter department ("Return" to search all depts, * to list depts):
Enter site ("Return" for local site, * to list all sites):
```

A local query for people called barker would be specified thus:

```
Enter the person's name (or "?" for help, "q" to quit): barker
Enter department ("Return" to search all depts, * to list depts):
Enter site ("Return" for local site, * to list all sites):
```

The results would be displayed thus:

```
name:           T Barker
department:     Finance Division
phone:          2553

name:           Paul Barker
department:     Computer Science
phone:          7366
email:          P.Barker@uk.ac.ucl.cs
```

6.3 Searching Further Afield

Substring matching is used to simplify the specification of queries. Searching can be done on up to 5 organisations from one query. This limit was arbitrarily chosen to curtail expansive searching. The behaviour of the interface can be seen from the following query and set of results:

```
Enter the person's name (or "?" for help, "q" to quit): smith
Enter department ("Return" to search all depts, * to list depts): comp
Enter site ("Return" for UCL, * to list all sites): ford

Searching site: Salford University Business Services Ltd...
No such department
Searching site: Salford University...
No such department
Searching site: Rutherford Appleton Laboratory...
No such department
Searching site: Bradford University...
```

```
  Searching dept: Computer Science...
  Searching dept: Computer Centre...
Searching site: Oxford University...
  Searching dept: Computing Service...

name:           Margaret Smith
department:     Computing Service
phone:          0865-...
organisation:   Oxford University

name:           Prof R A J Ord-Smith
department:     Computer Centre
phone:          0274 ...
email:          R.A.J.Ord-Smith@bradford.ac.uk
organisation:   Bradford University
```

6.4 The Appearance of the Results

Telephone numbers may be configured to appear as local extensions if appropriate, or omitting the country code if within the same country. Electronic mail addresses are by default in rfc822 domain order, but may be configured to U.K. grey-book order.

If more than 20 entries are returned, the results are given in a condensed format:

```
N Jones        2510          UCL Union
A Jones        82-9802       Surgery
T Jones        7077          Student Residences Office
T Jones        2338          Slade School of Fine Art
D Jones        82-5279       Physiology
C L Jones      7139          Physics and Astronomy
G O Jones      3468          Physics and Astronomy
P S Jones      3483          Physics and Astronomy
...
```

6.5 Quitting

Type “q” at the prompt for a person’s name.

Chapter 7

FRED

fred is a DUA optimised for White Pages queries, it is actually implemented as an interface to *dish*, hence the name FRED — FRont End to Dish.

7.1 Giving Commands to Fred

After invoking *fred*, you are prompted with "`fred> `" indicating that *fred* is ready.

If *fred* is invoked interactively, it will look for a file in your home directory called *.fredrc*. It will execute the commands contained in this file just as if you had typed them directly to *fred*. Following this, you are given the "`fred>`" prompt.

7.2 Let your Fingers do the Walking

Although *fred* has several commands, the most interesting command is `whois`, which performs a white pages query.

Let's begin with some simple examples and introduce the other commands along the way. If you already know the handle of the person you're interested in finding out about, just give the handle:

```
fred> whois @c=US@cn=Manager
Manager (1)

Handle:         @c=US@cn=Manager
```

7.2.1 The Alias Command

Since handles are long strings, *fred* will automatically maintain a list of aliases of the entries you have seen in the current session. The alias is always a number. When an entry is displayed, it appears on the first line in parenthesis after the name of the object. In the example above, the alias is `1`.

To find out what aliases are currently defined, use the **`alias`** command:

```
fred> alias
1    @c=US@cn=Manager
```

Thus, the previous **`whois`** command could have been shortened to simply:

```
fred> whois !1
Manager (1)

Handle:        @c=US@cn=Manager
```

Each time you invoke *fred*, its list of aliases is empty. If there are few handles which you use often, you might wish to define them in your *.fredrc* file, e.g.,

```
alias "@c=US@o=DMD@cn=Manager"
```

Of course, the ordering of aliases is important. *fred* will start numbering from 1 starting with the first **`alias`** command.

7.2.2 Back to Searching

Suppose however, that you don't know the handle for the person. In this case, you need to specify some search parameters. Logically, the first step is to ascertain the organization which the person is likely to be associated with, e.g., "Performance Systems International". This is done as:

```
fred> whois organization psi
Performance Systems International (1)  +1 800-836-0400 (Operations)
    aka: PSI

PSI Inc.
  Reston International Center
  11800 Sunrise Valley Drive
  Suite 1100
  Reston, VA 22091
  US

PSI Inc.
```

```
  5201 Great American Parkway
  Suite 3106
  Santa Clara, CA 95054
  US

PSI Inc.
  165 Jordan Road
  Troy, NY 12180
  US

Telephone: +1 800-836-0400 (Operations)
           +1 800-82PSI82 (Sales)
           +1 703-620-6651 (Corporate/Reston Office)
           +1 518-283-8860 (Troy Office)
           +1 408-562-6222 (Santa Clara Office)
FAX:       +1 703-620-4586
           +1 518-283-8904
           +1 408-562-6223

value-added provider of networking services

Locality:    Reston, Virginia

Name:     Performance Systems International, US (1)
Modified: Mon Jul 30 05:18:24 1990
      by: Manager, Performance Systems International,
            US (2)
```

Second, to search for a particular person, you might use:

```
fred> whois rose -area 2
Marshall Rose (3)                                    mrose@psi.com
     aka: mtr
     aka: Marshall T. Rose

Principal Scientist
PSI, Inc.
  POB 391776
  Mountain View, CA 94039
  US

PSI, Inc.
  5201 Great American Parkway
  Suite 3106
  Santa Clara, CA 95054
  US

Telephone: +1 415-961-3380
```

```
            +1 408-562-6222 x6221
FAX:        +1 415-961-3282
            +1 408-562-6223

Mailbox information:
  internet: mrose@psi.com
  internet: mrose@cheetah.ca.psi.com

Principal Implementor of the ISO Development Environment

Beleaguered Manager of the PSI White Pages Pilot Project

A savvyNerd according to noSauce

Locality:    Santa Clara, California

Drinks:       Iced Tea (and lots of it...)

Name:     Marshall Rose, Mountain View,
            Research and Development,
            Performance Systems International,
            US (3)
Modified: Mon Sep 24 14:43:36 1990
      by: Manager, US (5)
```

Note the use of the alias 2. The command could also have been:

```
fred> whois rose -area "@c=US@o=Performance Systems..."
    ...
```

Double-quotes are used so that the DN appears as a single token to *fred*.

Of course, this two-step process, whilst logical, is tedious. Thus, you can combine them like this:

```
fred> whois rose -org psi
    ...
```

which says to look for any organizations with "psi" in its name. Then, for each of these, look for something called "rose".

7.2.3 The Area Command

Suppose you want information on several persons belonging to an organization. You can use the `area` command, by itself, to tell *fred* where to search for subsequent commands. For example,

```
fred> area "@c=US@o=Performance Systems International"
```

or simply

```
fred> area 2
```

both tell *fred* the default area used by the `whois` command. Of course, you can still use the '`-area`' area with the `whois` command to override the default area. Thus,

```
fred> whois alan -area "@c=US@o=Columbia University"
```

will do what you expect.

If you use the `area` command without any arguments, *fred* will tell you what its default area is:

```
fred> area
@c=US@o=Yoyodyne
```

This indicates the default area for all commands, *including* any subsequent `area` commands. Thus, issuing:

```
fred> area @c=US@o=Yoyodyne
@c=US@o=Yoyodyne

fred> area ou=Research
@c=US@o=Yoyodyne@ou=Research
```

is equivalent to

```
fred> area @c=US@o=Yoyodyne@ou=Research
@c=US@o=Yoyodyne@ou=Research
```

because a leading "`@`"-sign was not used before `ou=Research`.

As you might expect, there is a special string "`..`" which may be used to move up one level:

```
fred> area ..
@c=US@o=Yoyodyne
```

Combinations are possible as well, such as:

```
fred> area ..@"o=Performance Systems International"
@c=US@o=Performance Systems International
```

which moves up a level and then down to
`o=Performance Systems International`

7.2.4 Getting Help

For a brief summary of *fred* commands, type:

```
fred> help ?
```

This will list the commands that *fred* knows about along with a one-line summary of their function.

For help on a particular command, type the name of the command followed by '`-help`', e.g.,

```
fred> alias -help
```

If you need more help, try

```
fred> manual
```

which is the same as

```
% man fred
```

from the shell.

7.2.5 Quitting

To terminate *fred*, simply use:

```
fred> quit
```

7.3 Advanced Usage

This chapter has given a very brief overview of the basic Fred commands, for full details you should consult [MRose90a], which tells you how to make more complex search requests, edit your own entry and how to use Fred to compose mail addresses using the MH mail system.

Chapter 8

Pod

POD is a Directory User Agent developed for the X Window System version 11 release 4, using the Athena widget set. It is intended to be an interface for the "naive" user. If you want to use the full power of X.500 you should use the DISH interface described in Chapter 4 of this manual.

8.1 Types of Widget

POD makes use of various simple on-screen devices called widgets. The most important of these are described in the following sections.

8.1.1 Buttons

A button is a rectangular screen area which when selected will activate some specific function. A button is selected by pointing at it with the cursor and then clicking with the first mouse button.

Most buttons contain a label and sometimes graphics which denote the button's function. In addition POD buttons are denoted by a change to a pointing hand cursor or by a border becoming highlighted.

A button box is simply a collection of buttons.

8.1.2 Dialogue Boxes

A dialogue box is a means of supplying textual information to an application, and can be thought of as a mini text editor. The dialogue boxes used in POD follow the command structure of the EMACS editor.

Dialogue boxes, like most others widgets, only become active (or have the input focus) when pointed at by the cursor.

8.1.3 Menus

A menu is a vertically arranged collection of buttons that appears on selection of a menu button.

8.2 Using POD

POD is invoked with the following command:-

```
pod [-t <tailor file>] [-T <oidtable>]
    [-c <dsa address>] [-u <quoted username>] [-p <password>]
```

`-t` is used to tell pod which tailor file to use in place of the default system dsaptailor file.

`-T` is used to force POD to use an alternative oidtable.

`-c` is used to bind to a DSA other than the local default.

`-u` is used to bind as a specific user.

`-p` is used to bind against the given password.

POD is made up of a number of separate windows, described in the following sections.

8.2.1 The Main Window

POD's main window is comprised of three sections: a current position display, a search input box and a button box. Each of these is described below.

Current Directory Position

All operations are performed relative to a base directory entry. The name of this entry is displayed in the main window under the title "Current Directory Position".

Parts of the displayed entry name can be selected in order to move to other positions in the directory.

Searching for Entries

The search input area contains a dialogue box and a menu button.

The dialogue box is used to enter a value describing an entry to be searched for, for example if searching for the entry for "Damanjit Mahl" a suitable value would be "D Mahl".

The menu button specifies the type of entry being searched for (the search type), in the above example this would be "Person". The search type can be changed by pulling down the menu attached to the type button and selecting from the contained list of types.

Searches are activated by clicking on the "Search" button or by pressing the RETURN key.

Main Functions

The buttons displayed in POD's main window are:

- Quit - Quit from POD
- Help - Invoke interactive help
- Search - Search for specified entry
- List - List entries under current position
- History - Display history of visited entries

8.2.2 The List Window

A list window displays a list of named directory entries, which can be returned as a result of list, search or history operations. It comprises a pair of buttons, a list display and two message bars.

The upper message bar contains the information regarding the source of the displayed list, e.g., "Result of List under Brunel University". The lower message bar contains errors and constraint messages.

List Display

To move to or read a listed entry simply click on that entry's name. Non-leaf entries cause a move and read, whereas leaf entries only cause a read.

Close and Keep Buttons

The close button can be used to close the list window. Note that this does not iconify the window but rather deletes it and all contained information.

The keep button can be used to make the information contained in the list window semi-permanent. In the default case, results from ensuing list/search operations overwrite any "unkept" list windows, thereby limiting the number of windows containing temporary information to one. If the keep button is selected, displayed information becomes semi-permanent, and is only removed if the close button is selected. The text in a keep button changes from "Keep" to "Kept" upon selection, in order to distinguish from list windows containing transient data.

The List window that displays the session history does not have a keep button.

8.2.3 The Read Window

The read window displays selected parts of a directory entry. It is comprised of a message bar, a button box and a text window.

The message bar displays the entry's name. If selected this bar places the complete distinguished name of an entry into the primary X cut buffer. Ordinarily this is of little use, though can help when modifying attributes that require a distinguished name as a value.

The text window displays the body of the entry, which currently contains textual information and may contain a (single) fax image stored in the photo attribute.

It is possible to cut text from the text window.

The close and keep buttons perform a similar function to those described above for the list window.

The modify button activates a window containing facilities to modify the entry being read.

8.2.4 The Modify Window

The modify window allows the user to modify the attributes of an entry. Each attribute value displayed in the text window is contained in its own dialogue box. Pointing to a dialogue box will allow editing of the contained value.

Each value has an associated menu button (denoted by the menu icon) which allows various operations on a particular value, e.g. delete value, undo changes etc. In addition each attribute label, e.g., “commonName”, is itself a menu button and allows various operations on all values associated with that attribute, e.g. delete all values, undo all changes to this attribute, add a new value field. Undo operations only undo all changes since the last successful modify operation or else return all affected values to their original states.

The close and keep buttons behave consistently with those described for the read and list windows.

The modify button attempts to make the modification entered into the text window. A message reporting the success (or otherwise) of a modification request is displayed in the lower right message window.

8.2.5 Error and Message Popups

From time to time, during or after directory operations, POD will provide error or status reports in a popup that appears in the top left corner of the screen. Errors sometime require the user to click on the error window before normal operation can resume.

8.3 Configuration of POD

POD can be configured on a system-wide or per-user basis. POD installs system default files into a directory called *xd/duaconfig/* under ISODE's ETCDIR.

Per-user configuration files other than *.podrc*, must be held under a directory *.duaconfig/* contained in a user's home directory.

The POD configuration files are:

```
<CONFIGDIR>/readTypes,
<CONFIGDIR>/duaconfig/friendlyNames,
<CONFIGDIR>/duaconfig/filterTypes/Type_*,
<CONFIGDIR>/duaconfig/typeDefaults.
```

8.3.1 The .podrc file

The *.podrc* file is analogous to the *.quipurc* file for DISH (see Section 4.4.1), though less extensive in the number and flexibility of options provided. It has the following format:-

`flag: value`

The following flags are currently recognised:-

username: The name of the user to bind as.

password: The password to use when binding. For this reason care should be taken to ensure your *.podrc* file is not publicly readable.

service: A list of default service control flags (as defined in Section 4.3).

dsap: This can be followed by one of the dsaptailor options described in Section 13.2.

isode: This can be followed by one of the ISODE tailor options described in *Volume One* of this manual.

history: This can be followed by a number that specifies the number of entry names to be maintained in the history buffer.

prefergreybook: For those who prefer grey book mail address format. This takes values of TRUE or FALSE.

readnonleaf: This tells POD whether or not to read non-leaf entries. Values are TRUE or FALSE.

8.3.2 The readTypes file

The file *readTypes* contains a list of attributes that are to be read for entry display. The format of this file is;

```
"quotedAttributeName" numericOid
```

an example line then being

```
"photo"              0.9.2342.19200300.100.1.7
```

This format was chosen as it is the same format as the output from the OIDDUMP utility provided with ISODE.

8.3.3 The friendlyNames file

This file maps attribute names onto user-friendly names, for use in displaying the "Current Directory Position". The current defaults map the names onto empty strings so that the bare values are shown, e.g;

```
The World
GB
Brunel University
```

The format of the file is:-

```
attribute list : friendly name
```

Where an attribute list can be one or more comma separated attribute names.

8.3.4 The filterTypes files

POD searches are based upon a complex filter, e.g. the default filter for the type *Person* is:-

```
objectClass=person AND (cn~=* OR sn~=* OR title~=*)
```

Where represents a value supplied at search time.

The directory *filterTypes/* contains a set of files, each with a prefix *Type_*, describing the set of such abstract types used in POD. The set of contained files may be edited or added to.

The syntax used in these files is shown in the example below:-

```
<filter_type> ::= <filter_name> <filter>
<name> ::= "name:" <asciistring>
<filter> ::= <filter_item> | <assertion>
<assertion> ::= "(" <filt_type> <filter> <filter>
                <filter_list> ")"
<filter_list> ::= <filter> <filter_list> | <filter> | NULL
<filter> ::= <filter_item> | <assertion>
<filter_item> ::= "(" NUMERIC_OID <match_type> <value> ")"
<filt_type> ::= "&" | "|"
<match_type> ::= "=" | "~=" | "%="
<value> ::= "*" | STRING
<comment> ::= "#" STRING
```

Where the symbol “ =” represents approximate match, “%=” represents substring match and “=” represents exact match. The “*” character for value represents a value supplied at search time.

As an example, the file representing the default type *Person* is:-

```
#Composition of type Person
name:"Person"
( & (2.5.4.0="2.5.6.6") # objectClass = person
    ( | (2.5.4.3~=*) # cn (common name)
(2.5.4.4~=*) # sn (surname)
(2.5.4.12~=*))) # title
```

8.3.5 The typeDefaults file

The *typeDefaults* file defines the relationships between each of the type defined in the *filterTypes* directory. This is best described by the following example line from the provided *typeDefaults* file.

```
2.5.4.10:Person, Place, Department: Person
```

The first field defines the type of entry to which this line applies, the OID shown here is the one for *organizationName*. The second field lists the POD search types which are available when visiting an entry of the specified type. So this line specifies that the types *Person*, *Place* and *Department* are available when visiting an entry of type *organizationName*. The third field defines the default search type to be used when visiting an entry of the specified type.

Chapter 9

Xd

Xd is a Directory User Agent (DUA) for the X Window System which provides access to the OSI X.500 Directory. It was developed (along with Xdsm, and an early version of Pod) as a prototype DUA to stimulate feedback for the design of full DUAs for X and Microsoft Windows.

9.1 Starting Xd

Xd accepts the usual command line paramters for X applications. These are described in the online manual pages (X(1)). Xd also recognises the following options which relate to the QUIPU directory services:

```
-t tailor file
-c dsa name
-T oid table
-D directory
```

If Xd has not been installed, it is necessary to load the X resources file into the server. The following command in the Xd directory should achieve this:

xrdb –merge Xd.ad

If you change any of the resources in the Xd.ad file, you will need to load them into the server again. The –load option to xrdb overwrites all resources already in the server, but may be useful when a resource is to be removed. *xrdb –query* lists the current contents of the server's resource database.

9.2 The Components of Xd

The components which make up Xd are described below:

Quit

Exits Xd

Help

Pressing the Help button invokes a popup window comprising a Quit button and a help text window. The help displayed in the text window is dependent on the position of the mouse. So entering the Type button area will display a help page describing the Type button function. Entering the Help button restores the General Help page, and pressing Help restores the popup to its position just above the Help button. The Quit button on the Help popup removes the popup. The Help popup does not stop other Xd buttons operating.

Search Area

The Search Area shows the current directory position. This is the base position from which Lists and Searches are performed.

Search For

This shows the current string being searched for. All keyboard input is focused on this widget. It accepts most EMACS-like commands for line editing, and carriage return starts a search.

Type

The Type area has a button which invokes a menu of possible search types, and a text area showing the current search type. The search type specifies a search filter which will be used for searches. When the current directory position is changed, default search type and search type options are set. The filters and defaults are configurable. This is described in the manual entry for Pod.

Read Area

All directory reads, and any errors are displayed in this text widget. Photographs can be displayed automatically using Xphoto, and can be destroyed by clicking on the photograph.

Search/List Buttons

The search and list buttons should be quite obvious. Search performs a search using the string and type specified. List shows the children of the current position.

Widen

Widen makes the current position one stage less specific.

Look back

The look back button invokes a popup with a list of the last ten places selected from the list area. Clicking on an item in the list, makes that the new current position. The popup is only large enough to contain the longest directory position currently in the list. If a place is added to the list whilst it is still popped up, the popup does not resize.

List Area

The final area in Xd is the list area. This is used to display the results of lists and searches. Selecting an item from the list area causes Xd to try and read the entry and make the entry the current position. Making a leaf entry the current position is meaningless, as it will have no children and lists or searches will be pointless. Such is life!

9.3 Driving Xd

To "press" a button, move the cursor (that funny hand shape) and click (press and release) button one (usually the left hand button) on the mouse. To select an item from a list, point the cursor at the item you want to select and click button one on the mouse.

To scroll a read or list area, move the cursor into the scrollbar and press button two on the mouse. Now move the mouse up and down or left and right until the required screen area is displayed. Now release the mouse button. The thick horizontal lines with the black squares at one end allow the relative sizes of the bounded areas to be changed. Grab the line by pressing a mouse button on then black box (handle), move the mouse, then release the button.

9.4 Configuration

The configuration of search filters was mentioned earlier in the context of the Type button and menu. This configuration is common between Pod and Xd. A directory called duaconfig (under *Xd/Xd* in the source tree) contains example configuration files. A description of these configuration files is given in the Chapter on Pod (Chapter 8). Note however that there are a few differences.

- The *friendlyNames* file is Pod specific.
- Xd uses *.quipurc* not *.podrc*.
- Xd only recognises username, password, isode, dsap and service fields of *.quipurc*

Chapter 10

Attribute Syntaxes

All attributes recognised by QUIPU have an associated "string" representation which is used to store the data in the DSA. This chapter describes all the currently recognised syntaxes.

For most of the syntaxes, a BNF description is given, using the following base description:-

```
<a> ::= any of the 52 upper and lower case IA5 letters
<d> ::= any IA5 digit 0-9
<k> ::= any of the 52 upper and lower case IA5 letters,
              IA5 digits, and "-" (hyphen)
<p> ::= any IA5 character in ASN.1 PrintableSting
<CRLF> ::= IA5 Newline
<letterstring> ::= <a> | <a> <letterstring>
<numericstring> ::= <d> | <d> <numericstring>
<keystring> ::= <k> | <k> <keystring>
<printablestring> ::= <p> | <p> <printablestring>
```

The BNF description of attributes can be found in Appendix B, but this does not repeat the BNF used for syntaxes given below.

10.1 Standard Syntaxes

This section describes the Attribute Syntaxes defined by X.500.

There are many attributes that are represented by a sequence of printable characters. There are various was in which the contents of the string is restricted, each is represented by a different syntax, which are described below.

10.1.1 PrintableString

Standard Attributes	QUIPU Attributes
serialNumber	execVector

The printable string characters are:

`A` through `Z`
`a` through `z`
`0` through `9`
`'` (apostrophe)
`(` (left parenthesis)
`)` (right parenthesis)
`+` (plus-sign)
`,` (comma)
`-` (hyphen)
`.` (period)
`/` (solidus)
`:` (colon)
`?` (question-mark)
space

The value can be any character listed above, with matching as for CaseExactString.

When matching, multiple white spaces and tabs are treated as a single space character.

Approximate matching is supported for this syntax.

The only standard attribute of this type is `serialNumber`, this is used in the `device` objectClass, and represents the serial number of the device. QUIPU defines an `execVector` attribute which is used by the *iaed* program as the vector to pass to a program when starting it.

10.1.2 CaseExactString

No attributes currently use this syntax. Matching is as for PrintableString.

The value for this syntax can be one of two types, either a printable string, or a T.61 string represented as follows:

```
<StringValue> ::= "{T.61}" <T61String> | <printablestring>
<T61String>   ::= Any character defined to be in T.61 String.
```

If a T.61 string is used, where possible the character is displayed using the equivalent ASCII character. Characters in T.61 string that can not be represented by an equivalent ASCII character are quoted, using "`\xx`" where "`xx`" is the hexadecimal value of the character.

However, if your terminal has the ability to display characters using an ISO 8859-1 font (for example some of the X Window System fonts) then a large number of the T.61 characters can be displayed (and so are not quoted in hex!). You will need to tell the DUA that you have the ability to display the ISO 8859-1 characters. This can be achieved in one of three ways:

- In the system wide dsaptailor file, you can add

  ```
  ch_set ISO8859
  ```

  or

- In your .quipurc file you can add

  ```
  dsap: ch_set ISO8859
  ```

  however not all the DUA interfaces read this files (dish does!) or

- set the environment variable CH_SET to "ISO8859", e.g., using "csh"

  ```
  setenv CH_SET ISO8859
  ```

To enter T.61 characters is a little more tricky. In T.61 accented characters are represented by two octets, the first indicating the accent and the second the base character to be accented. Note that some combinations of accent and character do not have an equivalent in ISO 8859-1, and hence cannot be displayed on an ISO 8859-1 device. Table 10.1 shows some of the characters that can be represented, for the accented characters only the first octect is shown, the second octect can typically be any character, "o" is used in most of the examples.

For instance, to enter the T.61 string "Galápagos Penguin", you should use

Hex code	Description	Character
c1	grave accent	ò
c2	acute accent	ó
c3	circumflex	ô
c4	tilde	õ
c5	macron	ō
c7	single dot	ȯ
ca	ring	å
cb	cedilla	ç
cc	underscore	$\underline{o}$
cd	umlaut	ö
cf	caron	ǒ
a1	inverted exclamation mark	¡
s3	pound sign	£
a4	dollar sign	$
a6	hash	#
a7	paragraph sign	§
ab	left quotation mark	”
b0	ring	å
b1	plus/minus	±
b5	greek letter mu	μ
b6	pilcrow	¶
b8	division sign	÷
bb	right quotation mark	“
bc	quarter sign	$\frac{1}{4}$
bd	half sign	$\frac{1}{2}$
be	three-quarters sign	$\frac{3}{4}$
bf	inverted question mark	¿
e0	greek letter omega	Ω
e1	AE	Æ
e8	L stroke	Ł
e9	O stroke	Ø
ea	OE	Œ

Table 10.1: T.61 Character Codes

```
{T.61}Gal\c2apagos Penguin
```

The acute accent is represented by the "\c2" and appears over the following character — "a" in this case.

10.1.3 CaseIgnoreString

Standard Attributes
knowledgeInformation
commonName
surname
localityName
stateOrProvinceName
streetAddress
organizationName
organizationalUnitName
title
description
businessCategory
postalCode
postOfficeBox
physicalDeliveryOfficeName

RARE Attributes
userid
textEncodedORaddress
info
favouriteDrink
roomNumber
userClass
host
documentIdentifier
documentTitle
documentVersion
documentLocation
durName
wkdName
protocolProfile
objectID
friendlyCountryName

The syntax is the same as CaseExactString, except that when matching "characters that differ only in their case are considered identical".

Approximate matching is supported for this syntax.

The use of some of the attributes is now described, the more obvious ones are omitted.

knowledgeInformation: A description of knowledge mastered by a DSA.

localityName: Used to identify the geographical area or locality in which the object is physically located e.g.,

```
London
```

stateOrProvinceName: describes the state in which the `locality` is found; e.g.,

```
New York
```

title: An objects job title, e.g.,

```
Technical Manager
```

businessCategory: describes the business of the object, e.g.,

```
networking
```

This is used to find people sharing the same occupation.

physicalDeliveryOfficeName: describes the geographical location of the physical delivery office which services the postal address of this object; e.g.,

```
Troy
```

friendlyCountryName: A "nice" name for countries, as opposed to the two letter codes enforced by use of `CountryName`, for example

```
Great Britain
```

10.1.4 CaseIgnoreList

No standard attributes currently use this syntax.

The CaseIgnoreList syntax consists of a sequence of CaseIgnoreString values as shown by the BNF below.

```
list = <list_component> | <list_component> "$" <list>
list_component = [ "{T61}" ] <string>
```

When the list is displayed to a user, the "$" is replaced with a newline. Ordering is preserved. For example the entry in an EDB file

```
caseIgnoreAttribute= this is a $ multi line $\
attribute definition
```

would be shown to a user as

```
caseIgnoreAttribute= this is a
                     multi line
                     attribute definition
```

QUIPU only allows equality matching for this syntax.

10.1.5 CountryString

Standard Attributes
countryName

This syntax is treated as a PrintableString, with matching rules as for CaseIgnoreString, and the restriction that the string must be one of the codes defined by ISO 3166.

10.1.6 IA5String

QUIPU Attributes
control
quipuVersion

RARE Attributes
aRecord
mDRecord
mXRecord
nSRecord
sOARecord
cNAMERecord

This syntax is handled as PrintableString, except a wider range of characters are recognised (i.e any character in IA5 string, characters such as NewLine are quoted using the same mechanism as described for T.61 string in Section 10.1.2), with matching rules as for CaseExactString.

10.1.7 VisibleString

RARE Attributes
nRSSystemDescription

This is currently treated as an ASN.1 attribute (see Section 10.6).

10.1.8 OctetString

No attributes use this syntax directly.

Characters that can not be printed as an ASCII are represented using the quoting mechanism described in Section 10.1.2.

10.1.9 NumericString

Standard Attributes
x121Address internationaliSDNNumber

The value is simply a numeric string (digits 0 through 9 only)

```
<NumericValue> ::= <numericstring>
```

The two attributes are used thus:

x121Address: defines the X.121 Address of an object as defined by the CCITT Recommendation X.121

internationaliSDNNumber: An International ISDN Number as defined by CCITT Recommendation E.164.

10.1.10 DestinationString

Standard Attributes
destinationIndicator

Behaves as a `<printablestring>`, with CaseIgnoreString matching rules.

The only attribute of the type `destinationIndicator` is used to define the addressee as required by the Public Telegram Service.

10.1.11 TelephoneNumber

Standard Attributes
telephoneNumber

RARE Attributes
homePhone

PSI Attributes
mobileTelephoneNumber pagerTelephoneNumber

The value should be a string describing the phone number of the object using the international notation; e.g.,

```
+1 518-283-8860
```

or

```
+1 518-283-8860 x1234
```

In general, the syntax is:

```
"+" <country code> <national number> [ "x" <extension> ]
```

Matching is as defined for CaseExactString, except that all space and "–" characters are skipped during the comparison.

10.1.12 PostalAddress

Standard Attributes
postalAddress
registeredAddress
homePostalAddress

```
<PostalAddressValue> ::= <address_component> |
                         <address_component> "$" <address>
<address_component> = <StringValue>
```

For example UCL $ Gower Street $ London. You are limited to a maximum of 6 <address_component>'s, each with a maximum of 30 characters. (Note each <address_component> can include T.61 strings — see Section 10.1.2 for details of this).

Please note: substring matching is not supported by QUIPU for this syntax.

10.1.13 DN

Standard Attributes
aliasedObjectName
member
owner
roleOccupant
seeAlso

QUIPU Attributes
masterDSA
slaveDSA
relayDSA

MHS Attributes
mhsMessageStoreName

RARE Attributes
manager
documentAuthor
secretary
lastModifiedBy
associatedName

```
<DNValue> ::= <name>
```

Distinguished names are discussed in Section 3.2.3, the BNF is repeated for completeness:

```
<DNValue> ::= <rdn> | <rdn> "@" <DNValue>
<rdn> ::= <attribute> | <attribute> "%" <rdn>
```

The DNs held by the attribute have the following meanings:-

aliasedObjectName: The name of an aliased object.

member: Specifies a member of a `groupOfNames`.

owner: The name of the person responsible for the associated object.

roleOccupant: The person that fills an organisational role.

seeAlso: Other objects that may be of interest.

masterDSA: The DSA holding the master EDB file.

slaveDSA: DSAs holding slave EDB files.

relayDSA: A DSA to relay operations to if your DSA is not connected to a needed network community. Section 14.2.3 describes this in more detail.

manager: The manager of a DSA.

documentAuthor: Author of a document!

secretary: The name of a personal Secretary.

lastModifiedBy: The object that last modified the referenced object!

associatedName: The DN associated with a DNS domain.

10.1.14 OID

Standard Attributes
supportedApplicationContext

MHS Attributes
mhsDeliverableContentTypes
mhsDeliverableEits
mhsSupportedAutomaticActions
mhsSupportedContentTypes
mhsSupportedOptionalAttributes

```
<OIDValue> ::= <oid>
```

All the syntaxes in this section are represented by OIDs as follows:

```
<oid> ::= <keystring> "." <numericoid> |
          <keystring> |
          <numericoid>
<numericoid> ::= <numericstring> |
                 <numericstring> "." <numericoid>
```

For example `2.3.4.2` or `attribute.6`.

In general, the value will be a string using oidtable mappings.

10.1.15 ObjectClass

Standard Attributes
objectClass

Although essentially an OID, a separate syntax is provided as an OID has additional semantics when used as an object class.

10.1.16 TelexNumber

Standard Attributes
telexNumber

```
<TelexNumberValue> ::= <printablestring> "$" <printablestring>
                       "$" <printablestring>
```

This is used to represent `number $ country $ answerback`, for example `007 28722 $ G $ UCLPHYS`.

10.1.17 TeletexTerminalIdentifier

Standard Attributes
teletexTerminalIdentifier

```
<TeletexTerminalIdentifierValue> ::= <printablestring>
              "$" <optstr> "$"  <optstr> "$" <optstr>
              "$" <optstr> "$" <optstr>
<optstr> ::= <printablestring> | (null)
```

Used to represent

```
terminal $ graphic $ control $ page $ misc $ private
```

10.1.18 FacsimileTelephoneNumber

Standard Attributes
facsimileTelephoneNumber

```
<FacsimileTelephoneNumberValue> ::= <printablestring>
                                    [ "$" <faxparameters> ]
<faxparameters> ::= <faxparm> | <faxparm>
                    "$" <faxparameters>
<faxparm> ::= "twoDimensional" | "fineResolution" |
              "unlimitedLength" | "b4Length" |
              "a3Width" | "b4Width" | "uncompressed"
```

For example `+44 602 123-4567 $ twoDimensional`

10.1.19 DeliveryMethod

Standard Attributes	MHS Attributes
preferredDeliveryMethod	mhsPreferredDeliveryMethods

```
<DeliveryValue> = <pdm_component> | <pdm_component> "$" <pdm>
pdm_component = "any" | "mhs" | "physical" | "telex" |
                "teletex" | "g3fax" | "g4fax" | "ia5" |
                "videotex" | "telephone"
```

For example `mhs $ telephone`.

10.1.20 PresentationAddress

Standard Attributes
presentationAddress

For more details see Section 14.2.1, *Volume One* of this manual and [SKill89a].

10.1.21 Password

Standard Attributes
userPassword

See Section 11.1 for a discussion of this attribute.

10.1.22 Certificate

Standard Attributes
userCertificate
cACertificate

```
<CertificateValue> ::= <Algorithm> "#" <EncryptedValue> "#"
           <Issuer> "#" <Subject> "#" <Algorithm> "#"
           <ProtocolVersion> "#" <Serial> "#" <Validity> "#"
           <Algorithm> "#" <EncryptedValue>

<Algorithm> ::= <OID> "#" <AlgorithmParameters>
<AlgorithmParameters> ::= | <IntegerValue> |
                            "{ASN}" <HexString>
<Issuer> ::= <DN>
<Subject> ::= <DN>
<ProtocolVersion> ::= <IntegerValue>
<Serial> ::= <IntegerValue>
<Validity> ::= <UTCTime> "#" <UTCTime>
<EncryptedValue> ::= <HexString> | <HexString> "-" <Digit>
```

10.1.23 CertificatePair

Standard Attributes
crossCertificatePair

```
<CertificatePairValue> ::= [<CertificateValue>] "|"
                           [<CertificateValue>]
```

At least one certificate should be present, although QUIPU will not enforce this.

10.1.24 CertificateList

Standard Attributes
authorityRevocationList certificateRevocationList

```
<CertificateListValue> ::= <AlgorithmValue> "#" <Issuer> "#"
                <AlgorithmValue> "#" <EncryptedValue> "#"
                <UTCTime> ["#" <RevokedCertificates>]

<RevokedCertificates> ::= <AlgorithmValue> "#"
                 <EncryptedValue>
                 [ "#" *(<RevokedCertificate> "#") ]

<RevokedCertificate> ::= <Subject> "#" <AlgorithmValue> "#"
                 <SerialNumber> "#" <UTCTime>
```

10.1.25 Guide

Standard Attributes
searchGuide

```
<GuideValue>   ::= [<objectClass> "#"] <Criteria>
<Criteria>     ::= <CriteriaItem> | <CriteriaSet> |
                   "!" <Criteria>
<CrtieriaSet>  ::= ["("] Criteria "@" CriteriaSet [")"] |
                   ["("] Criteria "|" CriteriaSet [")"]
<CriteriaItem> ::= ["("] <attributetype> "$" <matchType> [")"]
<matchType>    ::= "EQ" | "SUBSTR" | "GE" | "LE" | "APPROX"
```

Note the use of `@` for "and", as the `&` symbol has other meaning here. Some examples of Search Guide are:

```
Person # commonName $ APPROX
( organization $ EQ ) @ (commonName $ SUBSTR)
( organization $ EQ ) @ ((commonName $ SUBSTR) | \
                         (commonName $ EQ))
```

10.1.26 UTCTime

Standard Attributes
lastModifiedTime

```
<UTCTimeValue> ::= <printablestring>
```

The string should be formatted using the template `yymmddhhmmssz` where `yy` represents the year; `mm` represents the month; `dd` represents the day; `hh` represents hours; `mm` represents minutes; `ss` represents seconds; `z` represents the timezone.

For example the string `890602093221Z` is used to represent 09:32:21 at GMT, on June 2$^{\underline{\text{nd}}}$, 1989.

10.1.27 Boolean

No attributes use this syntax.

```
<BooleanValue> ::= <boolean>
<boolean>      ::= "TRUE" | "FALSE"
```

10.1.28 Integer

MHS Attributes
mhsDeliverableContentLength

```
<IntegerValue> ::= <d>
```

10.1.29 AccessPoint

QUIPU Attributes
subordinateReference
crossReference
nonSpecificSubordinateReference

```
<AccessPointValue> ::= <DN> "#" <PresentationAddress>
```

These attribute are used to give references to Non-QUIPU DSAs and are discussed in Section 14.6.

10.2 QUIPU Attribute Syntaxes

10.2.1 ACL

QUIPU Attributes
accessControlList

```
<aclvalue> ::= <aclwho> "#" <aclwhat> "#" <acltype> ["#"]
<aclwho> ::= "self" | "others" | "group #" <namelist> |
"prefix #" <namelist>
<aclwhat> ::= "none" | "detect" | "compare" |
                   "read" | "add" | "write"
<acltype> ::= "child" | "entry" | "default" |
                   "attributes #" <oidlist>
<oidlist> ::= <oid> | <oidlist> "$" <oid>
```

The use of ACL is discussed in Section 11.2.

10.2.2 Schema

QUIPU Attributes
treeStructure

Schema is currently represented by a single OID, and discussed fully in Section 14.7.

10.2.3 ProtectedPassword

QUIPU Attributes
protectedPassword

```
<ProtectedPasswordValue> ::= <StringValue>
```

The password encryption mechanism is encapsulated within the matching rules for this attribute; two values are "equal" if they are identically equal or if one is an encrypted representation of the other.

Note that "equals" is not transitive for this attribute: If $a = b$ and $b = c$, testing $a = c$ may give "incomparable" rather than "true" as the result. The reason for this is that in some circumstances properly testing this attribute for equality would consume an unacceptable amount of time. The security of the encryption mechanism depends on this!

10.2.4 SecurityPolicy

QUIPU Attributes
entrySecurityPolicy
dsaDefaultSecurityPolicy
dsaPermittedSecurityPolicy

This attribute is currently handled as "ASN.1", see Section 10.6.

10.2.5 EdbInfo

QUIPU Attributes
eDBinfo

```
<EdbInfoValue> ::= <name> "#" <name> "#" <namelist> ["#"]
```

This attribute is discussed in Section 14.10.

10.2.6 InheritedAttribute

QUIPU Attributes
InheritedAttribute

```
<InheritedAttributeValue> ::= [ <oid> "$" ]
                              [ "always" <InheritedList> ]
                              [ "default" <InheritedList> ]
<InheritedList> ::= "(" NEWLINE <AttributeSequence> NEWLINE ")"
```

This attribute is discussed in Section 14.7.5.

10.3 RARE Attribute Syntaxes

These syntaxes are also described in [SKill89c].

10.3.1 Mailbox

RARE Attributes
otherMailbox

```
<MailboxValue> ::= <printablestring> "$" <IA5String>
```

For example `internet $ quipu-support@cs.ucl.ac.uk`.

10.3.2 CaseIgnoreIA5String

RARE Attributes
rfc822Mailbox
domainComponent
nRSTextualDescription
associatedDomain

These attributes are handled as IA5Strings, but use the matching rules for the CaseIgnoreString syntax.

10.3.3 Photo

RARE Attributes
photo

Photos are a special case of "ASN.1" (see Section 10.6), but the output format is different. Your attention is drawn to Section 14.7.4 which discusses storing large attributes in a separate file.

With this version of QUIPU, the attribute should be encoded as a bit-string, in a two dimensional G3FacsimilePage encoding (as per Recommendation T.4). However, in the future this will migrate to an X.400 bodypart, containing a G3FacsimilePage, with the advantage this can the be one or two dimensional. The current photo decoders will recognise the new format.

10.3.4 Audio

RARE Attributes
audio

The data is in the form of a u-law encoded sound file. If you have a Sun Microsystems Sun 4 workstation, the demonstration "play" utility can be used to play the attributes (a modified version of "play" that ignores the file header is better — such as that made available by J. Michael Bauer of The University of Calgary, QUIPU-support can supply details if required).

This is an interim demonstration attribute, the format is temporary and may be replaced when a suitable standard audio encoding mechanism is found.

10.4 THORN System Attribute Syntaxes

Attribute	Syntax
thornACL	ThornACL
ruleDescription	RDType
objectDescription	ODType
attributeDescription	ADType
knowledgeReference	KnowledgeReference

The Syntaxes listed above are not recognised by QUIPU, and are thus handled as ASN.1 — see Section 10.6. However, QUIPU compatible syntax handlers for them are available as part of the THORN[FSiro88] project.

10.4.1 NRSInformation

THORN System Attributes
forwardOnlyInformation
reverseOnlyInformation
forwardAndReverseInformation

Although essentially a "Thorn" attribute, QUIPU does have a syntax handler for this syntax, however it is very complex and not defined here. It is defined in ??.

10.5 MHS Attribute Syntaxes

Attribute	Syntax
mhsORaddresses	ORAddress
mhsDLMembers	ORName
mhsDLSubmitPermissions	DLSubmitPermissions

The Syntaxes shown above are not currently recognised by QUIPU, and are thus handled as raw ASN.1 (see Section 10.6). However, the PP MHS System has QUIPU compatible syntax handlers that can be easily added to the *dish* program, and provides a tool for handling entries of the "distrbibution list" object class. PP is available under similar conditions to the ISODE, for details you should contact "PP-Support@cs.ucl.ac.uk".

10.6 ASN.1

As the preceding sections in this chapter have mentioned, not all syntaxes have a string representation defined by QUIPU, so are represented by the raw ASN.1.

An example would be:

```
photo= {ASN}0308207b4001488001fd...
```

where `0308`...is a hexadecimal representation (encoded using the "Basic Encoding Rules") of the ASN.1 defined attribute.

Attributes stored as ASN.1, will usually be matched correctly, with the following exceptions:

- There is an IMPLICIT Set in the ASN.1. The DSA will not detect the set, and so will not know to match components in arbitrary order.
- If special matching rules apply: for example, special rules to determine equivalence of telephone numbers. Such rules would need to be represented by code in the DSA.

Chapter 11

Introduction to Security Features

11.1 Passwords

When you bind to a DSA, to get write access you need to tell the DSA who you are — you do this by supplying your distinguished name. So that the DSA can authenticate you, you also need to supply your QUIPU password. Anyone knowing your password can act as you and alter your data. Therefore your password should be well-chosen and kept safe.

11.1.1 Choosing a Password

- Don't use the same password for QUIPU as you do for login.

  It is always bad practise (but convenient!) to use the same password on different systems. The manager of your QUIPU DSA can discover your QUIPU password. This doesn't affect the security of QUIPU (as your DSA manager can alter local data anyway), but the QUIPU manager is probably not allowed access to your login account.

- Don't choose an obvious word.

  Your password should **not** be any of the following:

  - Your name, initials, date of birth, place of work or similar personal details.

  - Any girl's name in any language.
  - The name of a film star, television personality etc.
  - The name of a character from a science-fiction or fantasy novel.
  - Computer jargon e.g., "foobar".
  - Any of the above, spelt backwards.
  - Any word from your system's on-line dictionary. Ideally, your password should not be a word at all (it could be the concatenation of two unrelated words, for example).

- Change your password from time to time.

11.1.2 Taking Care of Your Password

- Configuration Files

  It is possible to put your QUIPU password in a configuration file in your home directory (see the later Chapters on user interfaces). If you do this, you must make sure that the configuration file (.quipurc) is not publicly readable (by using the UNIX *chmod* command, for example).

- Access Control Lists

  Your QUIPU password is itself stored in QUIPU, as the *userPassword* attribute of your entry. You should make sure that the access control list for your entry grants public *compare* but not *read* access to this attribute. (The attribute must be publicly comparable so that QUIPU can check that have presented the right password when you start a session).

  The next section will explain how to set up access control lists.

- Don't give your password to other people.

  If several people need to be able to modify the same data, the access control lists can be set up so that this is possible without them sharing passwords.

- Don't write your password down!

11.2 Discretionary Access Control

"Discretionary Access Control" is any means by which users can (at their discretion) give other users access to data which they control. In QUIPU, access control lists are used to provide discretionary access control.

11.2.1 Model

Each node in the DIT held by a QUIPU DSA has a QUIPU access control list (ACL). The ACL is divided into three parts:

1. Child ACL

   This controls who may discover or change which entries are placed immediately below the node in the DIT.

2. Entry ACL

   This controls who may access the entry placed at the node.

3. Attribute ACL

   For every attribute of the entry, there is an Attribute ACL which controls who may access that attribute. To keep the representation compact, each entry has a default Attribute ACL. This is used for all the attributes of the entry that have not been explicitly given a different Attribute ACL.

Each Entry, Child or Attribute ACL (henceforth called an Object ACL) consists of a list of (access selector, access level) pairs. It associates every Distinguished Name with an access level, according to the following rule:

> Take every pair where the selector (left hand side) matches the name; The associated access level is the maximum of the corresponding right hand sides. If no selectors match the name, the associated access level is *none*.

The levels of access are as follows:

1. None

   No accesses to the object are allowed.

2. Detect

   A DUA can detect that the protected object exists.

3. Compare

   A filter (e.g. test for equality to some value) may be applied to the object.

4. Read

   The contents of the object may be read.

5. Add

   The contents of the object may be added to, but not removed from.

6. Write

   The contents of the object may be modified in any way.

The possible Access Selectors are as follows:

1. Entry

   Matches the entry itself only.

2. Other

   Matches everything.

3. Prefix <name>

   Matches <name> and everything below it in the DIT.

4. Group <name>

   Matches <name>. (This is not what the "group" selector was originally designed for, but it is currently what it does!)

The attribute syntax used to represent this is defined in Section 10.2.1, with the ASN.1 definition shown in Figure 11.1.

11.2.2 Detect Access

QUIPU treats the access level *none* as though it were *detect*. This means that is often possible to detect the presence of protected data, even if you have no access to it.

The reason for this is that it is very difficult for the Directory to pretend that data isn't there; carefully chosen queries can catch it out. Access control mechanisms that can be by-passed are very dangerous; they give a false sense of security. Accordingly, we have decided not to implement "undetectable data".

11.2.3 Effect of ACLs on Operations

This section explains which ACLs are checked for each of the X.500 operations.

1. List

   The Child ACL of the target must give at least *read* access. In addition, each child will only be listed if its Entry ACL gives at least *read*. Later versions of QUIPU may also require that the Attribute ACL of the distinguished (naming) attribute of the child give at least *read* access.

2. Search

   To search the immediate descendants of a node, that node's Child ACL must give at least *read* access. In addition, each child will only be searched if its Entry ACL gives at least *compare* access. The filter "present" may be applied to any attribute. Other basic filters evaluate to *maybe* unless the relevant Attribute ACL gives at least compare access.

   If an attribute within an entry does not have public read access, normally subtree searches on that attribute will fail. This is intentional, as it would be difficult to provide a consistent picture of the DIT, when unauthenticated DSP links are involved in the search. This restriction is lifted if, and only if, the entire subtree to be searched (using master or slave EDB files) is held within one DSA, and the association to that DSA is an authenticated DAP association.

3. Read

   The Entry ACL of the target must give at least *read* access and the Attribute ACL of each attribute read must give at least *read* access.

4. Compare

   The Entry ACL of the target must give at least *compare* access. The Attribute ACL of the tested attribute must give at least *compare* access.

5. Modify

   The Entry ACL of the target must give at least *add* access if attributes or values are to be added, and at least *write* access if they are to be removed. For each attribute changed, the Attribute ACL must give at least *add* access for an attribute or value to be added, and at least *write* access for an attribute or value to be removed.

6. Add Entry

   To add an entry below a node, that node's Child ACL must give at least *add* access.

7. Remove Entry

   To remove an entry, the Child ACL of its parent must give at least *write* access. No rights to the entry itself are required.

8. Modify RDN

   The Child ACL of the target's parent must give at least *write* access. If the operation needs to add an attribute (value), the target's Entry ACL must give *add* access and the Attribute ACL of the attribute must give *add* access. If the operation needs to remove an attribute (value), the target's Entry ACL must give *write* access and the Attribute ACL of the attribute must give *write* access.

11.2.4 Example Use of ACLs

A node representing a user might be given the following ACL:

- ChildACL is not applicable, and so omitted.

- EntryACL is {other, read} + {self, write} + {group=<Manager>, write}, so that only the user or a manager can change the entry. Using the ACL syntax, this is expressed as:

```
acl= other # read # entry
acl= self # write # entry
acl= group # <Manager Name> # write # entry
```

- DefaultAttributeACL is {other, read} + {self, write}, which leads to publicly readable attributes modifyable by the user. Using the ACL syntax, this is expressed as:

```
acl= other # read # default
acl= self # write # default
```

- The Attribute ACL for ACL is {other, read} + {group = <Manager Name>, write}, so that only the manager can change the ACL. Using the ACL syntax, this is expressed as:

```
acl= other # read # attributes # acl
acl= group # <Manager Name> # write # attributes # acl
```

- The Attribute ACL for Password is {self, write} + {other, compare} so that the user can change the password, DSAs can check the password, and only the user can read it.

```
acl= self # write # attributes # userPassword
acl= other # compare # attributes # userPassword
```

A node representing an organisation or organisational unit might be given the following ACL:

- ChildACL is {other, read} + {group=<Manager Name>, write}. Everybody can search the members of the organisation, but only the manager is allowed to add or delete members. This is represented as an attribute with:

```
acl= other # read # child
acl= group # <Manager Name> # write # child
```

- EntryACL is {other, read} + {group=<Manager Name>, write}. This is represented as an attribute with:

```
acl= other # read # entry
acl= group # <Manager Name> # write # entry
```

- DefaultAttributeACL is {other, read} + {group=<Manager Name>, write}. This is represented as an attribute with:

```
acl= other # read # default
acl= group # <Manager Name> # write # default
```

Every entry in the QUIPU DIT must have an acl attribute. If you do not supply one, the the default is added. The default ACL is often printed as

```
acl=
```

with no value. The default ACL is everybody read everything, but self can write, in long form this is expressed as:

```
acl= self # write # entry
acl= self # write # child
acl= self # write # default
acl= others # read # entry
acl= others # read # child
acl= others # read # default
```

For many entries this is sufficient.

It can be seen that this scheme gives a great deal of flexibility, without the addition of any protocol elements. The encoding is designed so that the volume overhead is not excessive for sensible access policies.

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 *REDE-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 directory in more detail.

Before you install QUIPU there are various compile time options you could consider setting which control the operation of QUIPU, inparticular 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*, *dsc*, *sd*,

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

pod, *ufn* and *xd*). 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 of both the DSA and 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_AVL`: 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 HAVE_PROTECTED`: If defined, enable use of protectedPassword attribute, and thus enable the use of protected simple authentication.

`#define [HAVE_RSA`: This option results in a DSA that can perform strong authentication, see Chapter 15 for details.

`#define COMPAT_6_0`: Operate in a manner compatible with QUIPU-6.0. To be fully compliant with X.500 distributed operations involving subtree searching across multiple DSA, this option should be removed. HOWEVER, it is essential for the pilot service that this option is SET until QUIPU-support indicate that is it safe to remove it!

`#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 to allow the use of the GNU "readline" package from the Free Software Foundation can be set in the *quipu/dish/Makefile*, details are given in the Makefile.

12.2 TURBO Options for Large Sites

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

The first option, TURBO_AVL, is used to speed the process of loading data from disk during startup (the TURBO_AVL option replaces the TURBO_LOAD option for this purpose), and to reduce DSA paging in many circumstances. With this option defined, code is enabled to keep each in core EDB in an AVL tree, instead of in a linked list. When QUIPU starts, it must check each entry that it loads against those sibling entries already loaded to ensure that each EDB file contains no duplicate RDNs. Normally, this is done by searching a linked list of loaded entries. The TURBO_AVL option causes AVL trees to be built and searched, making the loading process for big EDB files much faster. As an added benefit, fewer entries will be touched when resolving distinguished names, so DSA paging is reduced.

The second option, TURBO_INDEX, can be used to speed certain kinds of searches by building an index based on selected attribute types. There 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. NOTE: TURBO_AVL must also be defined in order to use TURBO_INDEX.

The third 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. NOTE: TURBO_AVL must also be defined in order to use TURBO_DISK.

To use the TURBO_DISK option, add a line like this to the *h/quipu/config.h* file:

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

You should also add a line like this 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 a gdbm file hierarchy. To use it, type "`tree2dbm database-directory`" where database-directory is the directory where the EDB file hierarchy begins. This script does a find starting in the specified directory for files named EDB and runs them through the *edb2dbm* program which creates a file called *EDB.gdbm*. The original EDB file is neither removed nor molested, so you'll need roughly twice the disk space. Alternatively, you can run *edb2dbm* by hand on each EDB file. The reverse operation (converting from a *gdbm* hierarchy to an EDB hierarchy) is done by the *synctree* shell script. It is a good idea to run *synctree* out of *crontab* once in a while to update the EDB file hierarchy.

Finally, it should be noted that parse errors will be reported somewhat differently in the *dsap.log* file with the TURBO_DISK option enabled. Since line numbers don't make much sense in a *gdbm* file, errors will be reported based on the Relative Distinguished Name of the offending entry. If you get a parse error (because of a non-printable character, for example), the best approach is to do something like this:

```
edbcat EDB >bob
```

Edit the file bob, locate the entry, fix the problem, then

```
edb2dbm bob
mv bob.gdbm EDB.gdbm
```

where this procedure assumes you are in the directory containing the bad EDB.gdbm file. The edbcat program can be found in the *quipu/turbo* directory and is used to convert from *gdbm* back to plain text EDB format.

12.3 Files

Regardless of how you install QUIPU and the ISODE, the number of files needed to run QUIPU are quite small.

In ISODE's `BINDIR` directory, typically */usr/local/bin/*, there are a few programs of interest:

dish: The DIrectory SHell
This is discussed in Chapter 4.

bind: Shell interface to *dish*
There are actually several links (listed below) to a program called *bind*. These act to export the *dish* interface to the UNIX shell. As such, you can issue commands to *dish* from the shell, rather than running *dish* directly.

add	compare
delete	dsacontrol
list	modify
modifyrdn	moveto
search	showentry
showname	squid

fred: A white pages user interface
See Chapter 7.

editentry: Edit a Directory entry
This is a simple shell script that *dish* invokes when you ask *dish* to edit an entry in the Directory.

unbind: Unbind from *dish*
This command is used to terminate *dish*.

In ISODE's `SBINDIR` directory, typically */usr/etc/*, the DSA resides:

ros.quipu: The QUIPU DSA
This program will be started once, for each DSA you are running, from rc.local. A script is provided to invoke this program, in case you need to restart it.

In ISODE's `ETCDIR` directory, also typically */usr/etc/*, there are a few programs and files of interest:

oidtable.at, oidtable.gen, oidtable.oc: These define the attribute types, generic object identifiers, and object classes known to the system. (An object identifier is a method used to unambiguously encode, among other things, the names of attributes and object classes.) These files you never deal with unless they are accidentally corrupted.

dsaptailor: This is the run-time tailor file for the DUAs. You will configure this file initially and then probably leave it alone.

isoaliases, isobjects, isoentities, isomacros, isoservices:
These are various databases used by the ISODE. These files you never deal with unless they are accidentally corrupted.

isologs: This script runs nightly under *cron*(8) to trim the ISODE log files, kept in ISODE's `LOGDIR` directory, typically */usr/tmp/*. This file you never deal with unless it is accidentally corrupted.

isotailor: This is the run-time tailor file for the ISODE. You will configure this file initially and then probably leave it alone.

Chapter 13

Configuring a DUA

It is suggested that you try to get a DUA operational by connecting to a "well known" DSA before you attempt to operate a local DSA. Or, if you have a DSA from a previous release of QUIPU — try and connect to that.

13.1 Connecting to a DSA

A DUA essentially only needs to know one thing to be able to contact a DSA, that is the OSI network address of the DSA. This parameter is defined in the file *dsaptailor*, together with some other parameters. The full set of parameters are described in Section 13.2.

The `dsa_address` parameter defines a local name and the network address of the DSA to initially contact. For example,

```
dsa_address  vicuna  Internet=bells.cs.ucl.ac.uk+50987
```

declares that the DSA locally referred to by the name `vicuna` is contacted by calling the network address `Internet=bells.cs.ucl.ac.uk`50987. The syntax of network addresses is discussed in briefly in Section 14.2.1 and more fully in *Volume Two* of this manual and [SKill89a].

As shown, the address is preceded by a private key `vicuna`. This can be used in some DUAs (including DISH) to specifying the address of the DSA to contact. If there are more than one `dsa_address` entries, the first entry will be used to supply the default DSA address.

A default dsaptailor file (taken from the *dsap/dsaptailor* file of the ISODE source tree) is installed as *dsaptailor* in the ISODE `ETCDIR` directory (usually

/usr/etc/) when the dsap library is installed, this supplies the addresses of various DSAs that you may be able to access.

To try to connect to one of the DSAs listed in dsaptailor, invoke *dish*, with a `-call` flag, e.g., "`dish -call giant`" will try to contact the DSA "giant" running at UCL. If the connection is successful, then the prompt "`dish ->`" will be returned. If the connection fails, the program will exit with an appropriate error message.

If this fails you might want to try connecting to some of the other registered DSAs, for example, try "`dish -call alpaca`", "`dish -call eel`" or "`dish -call anaconda`".

Many of these top level DSAs do not allow anonymous connections. If you see the message "`inappropriate authentication`" as a result of your connection attempt you will probably need to supply a DN using the "`-username`" flag to dish.

If you fail to contact a DSA at this point, there are likely to be lower level problems. You should turn up the ISODE logging (see *Volume Two* of this manual) to see what is happening to the network calls.

If you invoke *dish* without a `-c` flag (using the default *dsaptailor*), it will try to connect to the DSA defined by the first `dsa_address` entry.

dish is described in full in Chapter 4 of this manual.

13.2 Tailoring

The program configuration is tailored to allow you to change logging levels, and other parameters at run time. It is used by the QUIPU DUA procedures, and by the QUIPU User Interfaces.

The file *dsaptailor* is used for this purpose and consists of single value entries (e.g. oidtable), unless otherwise stated (e.g dsaplog). Each entry has a parameter followed by its value. The various options are:

`oidtable:` The path for the OID definition tables. NOTE: It is best to have this appear as the first entry of the tailor file, as other entries may contain attributes that need to be looked up in these tables There are three:

- *file.gen*, which contains generic names for building OIDs;
- *file.at*, which contains the OIDs for attributes; and,

- *file*.oc, which contains the OIDs for object classes.

For example,

```
oidtable    /usr/lib/quipu/OIDTable
```

will direct the DSA to consult:

```
/usr/lib/quipu/OIDTable.gen
/usr/lib/quipu/OIDTable.at
/usr/lib/quipu/OIDTable.oc
```

By default this variable is set to `oidtable` which refers to the tables *oidtable.** in the ISODE `ETCDIR` directory.

`dsa_address`: This parameter is described in Section 13.1.

`dsaplog`: Tailoring for the normal logging file. Each entry consists of one or more key/value pairs expressed as:

```
key=value
```

The keys are:

`file`: The name of the logfile.

`size`: The size in KBytes to which the logfile should be allowed to grow. When the log has reached this size, if the "zero" option below is set, then the log will be truncated, otherwise, no further logging will take place.

`level`: The levels of logging to be written to this log file. This can be any of the following levels:

`fatal`: fatal errors only

`exceptions`: serious, but hopefully temporary, errors

`notice`: general logging information

`trace`: program tracing

`pdus`: pdu tracing

`debug`: full tracing of events

`all`: log all events

For example to have all errors written to the file you will need

```
dsaplog level=fatal level=exceptions
```

`dlevel`: Do not log the specified log level, this is the opposite of the above entry.

`dflags/sflags`: The flags associated with the log may be set (with `sflag`) or unset (with `dflag`). The allowable options are:

`close`: close the log after each entry

`create`: create the log file if it doesn't exist

`zero`: truncate the file when it gets too big

`tty`: copy the logging information to the users tty

An example might be:

```
dsaplog level=notice size=30 file=quipulog dflags=close
```

This says log events at "notice" level, into the file "quipulog", do not let the file grow larger than 30Kbytes, and do not close the file after each logging message.

`stats`: Used to control the level of statistical logging (parameters as for `dsaplog` above).

`local_DIT`: The argument is a distinguished name. When some User Interfaces start, you will be automatically moved to this position in the DIT.

`oidformat`: Defines how object identifiers should be printed. Use one of:

```
oidformat   short
```

to print in short local key form, e.g.,

```
Country
```

or,

```
oidformat   long
```

to print in long object identifier form, e.g.,

```
joint.ds.attributeType.country
```

or,

```
oidformat   numeric
```

to print in numeric form, e.g.,

```
2.5.4.6
```

`photo`: The argument is has two parts, a "terminal type" such as "sun" or "xterm", the second is the name of the process that should handle displaying of photographs, for example

photo xterm Xphoto

tells the DUA to call the process Xphoto to handle photograph attributes if the user is on a terminal of type "xterm". Handling photographs is described more fully in Section 14.7.3.

`quipurc`: If the argument has the value `on`, then a program called *dishinit* will be run every time a user without a ".quipurc" file tries to access the directory. *dishinit* is discussed in Section 4.7.1.

`sizelimit`: Defines the maximum number of entries a successful list or search should return. For example,

```
sizelimit 20
```

sets the DAP default service control "sizelimit" to be 20 entries.

Chapter 14

Configuring a DSA

This chapter discusses how to configure a QUIPU DSA. We recommend that you get a DUA running before you try to get a DSA working.

14.1 Basic Formats and Structures

All of the information a DSA requires is stored on disk and is text structured. This includes various files (described later), and the local DIT database itself. A complete BNF description of the files is given in Appendix B on page 286.

14.1.1 Entry Data Block

A key component of the Directory database is the Entry Data Block, which is described fully in [SKill89b]. Figure 14.1 shows an example EDB file containing two "person" entries.

An EDB file contains a header, this is optionally but typically followed by a sequence of entries.

The header consists of two lines of text, the first must contain the string "`MASTER`", "`SLAVE`" or "`CACHE`", which indicates whether the data in the EDB file represents the authoritative `MASTER` data, a `SLAVE` copy of all the data, or some `CACHE`d entries.

The next line of the EDB is a string that describes the version of the EDB. Every time the EDB is altered, the version number should be changed, so that `SLAVE` EDBs elsewhere will be automatically updated. When a DSA

```
MASTER
19891025113003Z
CN= Colin Robbins
CN= Colin John Robbins
Phone= +44-1-387-7050 ext 3683
Surname= Robbins
Room= G10
Userid= crobbins
userClass= csstaff
rfc822Mailbox= C.Robbins@cs.ucl.ac.uk
Photo= {FILE}crobbins.photo
objectClass =thornPerson & quipuObject
acl=others # none # attribute # photo
acl=self # read # attribute # photo

CN= Steve Kille
CN= Steve E. Kille & Stephen Kille
Phone= +44-1-387-7050 ext 7294
Surname= Kille
objectClass = thornPerson & quipuObject
Room= G24
Userid= steve
userClass= csstaff
rfc822Mailbox= S.Kille@cs.ucl.ac.uk
```

Figure 14.1: Example EDB File

alters an EDB file, it writes the current (UTC) time in string format as the version string.

Generally following the header are a sequence of blank line separated entries. The concept of a "NULL EDB" file that contains just a header is allowed but discouraged. However it is sometimes useful as a temporary measure when creating a database.

An entry consists of a set of attributes, each attribute begins on a new line of the file. Section 3.2 discusses attributes in more detail and Chapter 10 describes the syntaxes used by all the attributes QUIPU recognises.

The first line of an entry is the Relative Distinguished Name (RDN) of the entry. The subsequent line contains the non distinguished attributes.

14.1.2 Object Class attribute

Of all the attributes an entry may have, the "objectClass" attribute is one of the most important from the configuration point of view. It defines the set of mandatory and optional attributes that must and may be present in the entry. For example, the object class "person" insists that there is a "surname" attribute, and there may optionally be a "telephone number" attribute.

QUIPU knows about all the standard object classes and attributes, some of those defined by the THORN project (see Section 16.2) and those defined by QUIPU itself (see Appendix C). The full set of object classes and attributes a DSA knows about is defined by the "oidtables" which are explained in Section 14.11.

An entry can belong to more that one object class. For example, an entry representing an organisation might have the following object class attribute:-

```
objectClass = organization & quipuNonLeafObject
```

And a person within that organisation might use the following object class definition:-

```
objectClass = organizationalPerson & quipuObject & thornPerson
```

Every entry in a QUIPU DSA should belong to either the "`quipuObject`" or "`quipuNonLeafObject`" object classes[1], as this allows attributes the DSA

[1] An entry may belong to "`ExternalNonLeafObject`" instead IF it is actually represented in the DIT by a non-QUIPU DSA — See Section 14.6.

needs to be added to an entry.

Further, object classes posses the notion of `class inheritance`. This means that an object class can be defined as a "subclass" of a previously defined object class with additional refinements. As a subclass, the newly defined object "inherits" all the semantics of its superclass, in addition to having additional semantics.

For example, the Directory defines an object class called `person`. This object class defines the attributes which a person in the real world might have. It may be useful to refine this somewhat to talk about persons who have network access. So, we need a new object class, e.g., `netPerson`. This can be defined in a straight-forward fashion:

> The object class `netPerson` is a subclass of the object class `person` which *may* contain an additional attribute, `netMailbox`.
>
> The syntax of an `netMailbox` is a simple string of printable characters which is not case sensitive when performing comparisons.

It is a QUIPU requirement that every entry that is not a leaf of the DIT should belong to the object class "`quipuNonLeafObject`".

This class has one mandatory attribute:

masterDSA: identifies the Directory entity which is responsible for maintaining the MASTER EDB for the children of this entry. The value is a Distinguished Name.

There is typically a single MASTER for a particular entry in the tree. Hence, this value is usually single-valued. When an entry is to be modified, the Directory must contact the entity responsible for the MASTER EDB for that entry in order to perform the modification.

This class has two optional attributes:

slaveDSA: identifies any Directory entities which have authoritative copies of the EDB for the children of this entry, and are prepared to resolve operations on that EDB file for a remote DSA. The value is one or more Distinguished Names.

treeStructure: identifies the object classes which may exist immediately below this entry. The value is one or more object classes. See Section 14.7.2 for full details of how to set this attribute.

Since a fundamental assumption of the Directory is that reads (queries) occur much more frequently than writes (updates), it is common to have several entities containing authoritative copies of an EDB. By keeping copies locally, queries can be answered with less latency.

14.1.3 Database Structure

All the local information held by a QUIPU Directory is held in an in-core database, this is loaded from disk when the DSA starts.

The data on disk is held in a UNIX tree of EDB[2] files that map the DIT. At every level in the DIT for which the DSA holds data, there is a single file called *EDB*. The top level of the DIT is stored in the UNIX directory defined by the `treedir` variable in the *quiputailor* file. For example, the setting

```
treedir     /usr/etc/quipu-db/
```

would define that the top level of the DIT would be found in this UNIX directory. If you hold a copy of the ROOT EDB file, it will be found here.

If an entry defined in an *EDB* file has children stored locally, then the *EDB* file for the children will be found in a sub-directory whose name is the string encoded Relative Distinguished Name of the entry. For example, underneath the ROOT, there are typically countries such as "c=GB". The data for this will be held in the file *c=GB/EDB*, or to give the fullpath name */usr/etc/quipu-db/c=GB/EDB*.

This mapping continues all the way down the DIT hierarchy, so for example, if an *EDB* file has an entry whose RDN is "ou= Computer Science", then if the entry for "ou= Computer Science" has sibling entries and these are stored locally, they can be found in the file *ou=Computer Science/EDB* relative to the directory that contains the EDB file with the "ou=Computer Science" entry.

NOTE that the case sensitivity of the sub-directory naming is one of the few areas within QUIPU where string matching is case sensitive. The case of the attribute type is taken from the definition of the attribute in the oid tables, whereas the case of the attribute value is the same as that found in the EDB file. Spacing is also important. There should be no spaces either

[2] Throughout this section the term "EDB" is used in the generic sense, and includes "EDB.gdbm" files if the TURBO_DISK compile option is used

side of the "=" sign, and only one space between each word contained in the name.

When an entry is modified, a new*EDB* file is re-written to disk. The old *EDB* is renamed *EDB.bak* to provide a limited back-up.

14.1.4 Long Distinguished Names

There is a problem with the above method for naming UNIX sub-directories with some versions of UNIX — particularly System V. In these systems directory names are limited in length. Some versions of UNIX will not allow space characters in files names (in any case they are hard to manage because of all the quoting required).

To allow for this, any distinguished name can be given a "mapping name", which will be used as the UNIX sub-directory name. For example the entry for "o=University College London", may be mapped onto the name "UCL" and thus stored in the file *UCL/EDB*. The mapping names are specified in a file called *EDB.map*. found in the same directory as the EDB file holding the entry to be mapped. So it the file *c=GB/EDB* contains an entry for "o=X-Tel Services Ltd", then the file *c=GB/EDB.map* may be used to map this onto "X-Tel".

The syntax of the *EDB.map* file is is:

```
<Distinguished Name> "#" <Mapped name>
```

so for the example used above the file will contain:

```
o=X-Tel Services Ltd#X-Tel
```

Only RDNs you want to map need to be in this file. If a name is not found in the mapping file, then the long directory name will be used.

When a DSA needs to create a sub-directory (e.g after an add operation) it will use the relative distinguished name for each subdirectory, unless the name is longer than the maximum number of allowed characters (usually 15). In this case the DSA will generate a shorter mapped name, and write this to the *EDB.map* file. A generated mapped name is based on the UNIX "mktemp" procedure call and produces names such as "XTelServia01950".

14.2 Setting up an Initial DSA

These instructions are assuming that you are trying to set up a DSA with the following characteristics:-

- It is the first DSA in an Organisation
- It is not the first DSA in the Country
- It holds a copy of the root EDB

This is found in the example:-

others/quipu/quipu-db/organisation

There are two other examples which might also be used as illustrations for national DSAs and organisational DSAs not holding a copy of the root EDB. These are in :-

others/quipu/quipu-db/national

and

others/quipu/quipu-db/non-root

Note that if you are going to be running a DSA in the United States, then you should skip this section and refer to the document [MRose90b], which is provided in the ISODE documentation set (look in the source tree area for the directory *doc/whitepages/admin/*). This document describes turn-key installation mechanisms for DSAs in the United States.

To start a DSA with one of these example databases go into the relevant database directory and type:-

```
$(SBINDIR)ros.quipu -t ./quiputailor
```

The `-t` flag tells QUIPU to use the tailor file *./quiputailor* rather than the default file (*quiputailor* in the ISODE `ETCDIR` directory).

This will cause the DSA to print some logging information onto the screen, followed by the message

```
DSA c=GB@cn=toucan has started on localHost=17003
```

```
cn=Toucan
presentationAddress= localHost=17003
edbinfo= #cn=giant tortoise#
description= Demonstration DSA
description= Bird with large colourful bill.
objectClass= quipuDSA & quipuObject
manager= c=GB@o=X-Tel Services Ltd@cn=Camayoc
acl= others # compare # attributes # userPassword
userPassword = toucan
quipuVersion= quipu 6.8 #3 (trellis) of Mon Feb 4 09:26:47 GMT 1991
supportedApplicationContext= QuipuDSP & X500DSP & X500DAP
```

Figure 14.2: Example DSA Entry

The default setup assumes you have TCP/IP access and starts a DSA on the IP address of the local machine (127.0.0.1). If you do not have TCP/IP, you will need to change the address the DSA will attempt to listen on, the next section delves into the world of addresses!

The presentation address of the example DSA is found in the file

others/quipu/quipu-db/organisation/c=GB/EDB

The first entry in this file is the entry for a DSA called "c=GB@cn=Toucan", which is the name of the DSA we are trying to start (as defined by the "mydsaname" entry in quiputailor). The entry is shown in Figure 14.2. The attribute `presentationAddress` defines the address that the DSA is going to listen to the network on. If you need to listen on a different address, you should change the value of the attribute to the address you want to listen on.

Having started your DSA you should be able to connect to it by invoking *dish*. If you are using the default DSA address, and are using the default *dsaptailor* file, then invoking *dish* without arguments is sufficient. If you are not using the default address or "dsaptailor" file, then you will need to edit *dsaptailor* in the ISODE `ETCDIR` directory. You should add an entry

```
dsa_address  toucan  <presentation address>
```

Note that `<presentation address>` should have *exactly* the same value as the `presentationAddress` attribute in the DSAs entry in the EDB file. Now to contact the DSA use

```
dish -c toucan
```

Once connected to the DSA, try issuing the command:-

```
list "@c=GB@o=University College London@ou=Computer Science"
```

You should get a list of four names back:-

```
1 commonName=Colin Robbins
2 userid=quipu%commonName=Colin Robbins
3 commonName=Steve Kille
4 commonName=Michael Roe
```

If this happens you have a working DSA. (The entry numbered 2, is an example of an RDN with multiple values !)

14.2.1 Presentation Addresses

This section is a brief introduction into presentation addresses, and may be all you need to know. For the brave, more details are given in *Volume Two* of this manual and [SKill89a].

The example address used in the previous section for the demonstration DSA was "`localHost=17003`". This is an address using the TCP/IP transport stack, and is the TCP loop back address. You should NOT use this is in any DSA involved in the pilot DIT for reasons explained at the end of the section.

If you want to use the TCP/IP transport stack (using RFC1006) there are two possibilites for an address. If you are connected to the Capital-I Intenet network, you can use addresses of the form

```
Internet=128.16.5.31+17003
```

"128.16.5.31" is the IP address of your machine, which can usually be found in the */etc/hosts* file on your system. You can use the DNS name of you host instead, but the ISODE will replace this with the IP address when possible to aid portability of addresses. "17003" is the TCP port number the DSA will listen on.

If you want to use TCP/IP on a local Ethernet[3], that is not connect to the Capital-I Internet, you will need to define an local network community,

[3] Ethernet is a trademark of the Xerox Corporation.

this process is described in full in *Volume Two* of this manual and should result in addresses of the form

```
LOCAL-ETHER=128.16.5.31+17003
```

Do NOT use Internet= addresses UNLESS you are connected to the Internet.

Now onto the X.25(80) community. If you have IPSS access, you can use addresses of the form

```
Int-X25(80)=23421920030045
```

where 23421920030045 is the DTE of you host. If you plan to have more than just a DSA (e.g., *iaed* or *tsapd*) listening on X.25 on the same machine, we advise the use of a two digit subaddress (45 in the example). The DTE should have the DNIC included, full details of tailoring DNIC's are given in *isotailor*(5n). The X.25 parameters PID and CUDF can be specifed if needed, see [SKill89a] for details.

If you do not have connectivity to the IPSS network, as in the TCP/IP case you will need to define a local community. The ISODE has knowledge of two such X.25 communites built in. In Europe, the IXI network should be addressed using an address of the form

```
IXI=20433450210398
```

In Great Britain, the Janet network should be addressed using an address of the form

```
Janet=00002100102998
```

Your DSAs address should contain a component for every network you have access to. Multiple components can be linked using the "`|`" symbol, for example

```
Internet=128.16.5.31+17003|IXI=20433450210398
```

Using this addressing mechanism it is possible to use Transport, Session and Presentation selectors, however in the context of a QUIPU DSA they are unnecassary, hence we advise against their use, again details can be found in [SKill89a].

Correctly defining your DSAs address is important. A paper [PBark89] describes the problem in detail. Briefly, X.500 was defined assuming a single global network, unfortunately the world is not yet like that, there are at least two major communities. QUIPU DSAs know about this, and make a careful choice between use of referrals and chaining in an attempt to make sure all operations suceed. If your DSA is incorrectly addressed, other DSAs may make the wrong assumption about your DSA, and so your DSA will not be able to contact certain parts of the DIT, and they will not be able to contact you. For a similar reason you should not use the "localHost" macro for addreses, as other DSAs will think they can connect to the DSA in question.

14.2.2 Choosing a Name for Your DSA

Every QUIPU DSA MUST have an entry in the DIT, hence your DSA will need a unique **distinguished** name. This entry is used by other DSAs to identify your DSA, and so is needed if other DSAs are going to be able to see your DSA. The examples are tailored to start a DSA called "toucan", and will be sufficient to get an example DSA started, but is not unique, so will not be of much use when you want to start adding your own data, and want to connect into the global DIT. NOTE the DN has to be unique, but not the RDN component, thus both "cn=Wombat" and "c=GB @ cn=Wombat" are allowed!

There are other two aspects to consider in choosing a name for your DSA.

Firstly, it is a QUIPU convention that DSAs should be named after endangered South American wildlife, and that the entry for the DSA should contain a description of the animal or plant in question. Some example animal names are shown in Table 14.1. A more comprehensive list can be found in the IUCN's "Red Book" [IUCN82]. This is not a just a game, there is a serious point. The authors believe it is wrong to name DSA "UCL.DSA" or similar, as this binds the DSA to the organistion too tightly, especially at the higher levels of the DIT.

Secondly, the entry for your DSA must be visible to other DSAs that do not know how to contact your DSA. So, the MASTER copy of your DSAs own entry must be held at least one level higher up in the DIT than the part of the DIT it holds as `MASTER` data. For example the DSA which holds "c=GB @ o=University College London @ ou=Computer Science", must be held at the "c=GB @ o=University College London" level (e.g. "c=GB @

o=University College London @ cn=wombat") or above. Such naming also helps prevent loops as described more fully in [SKill89b].

Typically, you will not hold the MASTER EDB file containing your DSA locally. However, you do still have control of the entry as if you held the MASTER copy. This is explained in more detail in Section 14.4.

In practice DSAs should be named fairly high up the tree. Each country should have at least two DSAs named at the root level. Each Organisation should have at least two DSAs named at the national level.

You should use *dish* to find out if the name you want is already taken. For example, if you are creating a DSA for an organization in Great Britain, you might use:

```
% dish -c "Giant Tortoise"
Welcome to Dish (DIrectory SHell)
Dish -> search @c=GB -filter objectClass=dsa -nosize
```

This will print out the list of names already in use.

14.2.3 Setting up YOUR DSA

Having chosen a name, you will need to tell your DSA its name, and make sure it can find an EDB file for its own entry.

A DSA finds it own name from the "mydsaname" variable defined in the file *quiputailor* (see Section 14.3 for details). An example `quiputailor` entry would be:-

```
mydsaname: "c=GB@cn=a dsa name"
```

Having read this name from `quiputailor`, a DSA will try to find the corresponding entry, in this case by looking in the *c=GB/EDB file*.

If you hold the EDB that the entry should be in, simply make sure the entry is in that EDB and then it will be found. If you do not hold, and do not want to hold the EDB in which your DSA is named (e.g., your DSA is called `c=GB@cn=toucan` but you do not hold the EDB `c=GB`), then you should (normally) supply a cached copy of the EDB which contains only the entry for the DSA, and not all the other entries the full EDB would have. If you do not supply it, your DSA will have to rely on others DSAs before it can start.

Agouti	Alpaca	Anaconda	Queen Anglefish
Anole	Carpenter Ant	Leaf Cutter Ant	Giant Anteater
Two toed Anteater	Antshrike	Aperea	Arapaima
Armadillo	Giant Armadillo	Hairy Armadillo	Six-banded Armadillo
Brazilian Three-banded Armadillo	Axolotl	Fruit Bat	Vampire Bat
Spectacled Bear	Long Haired Beetle	Passionflower Leaf Beetle	Rhinocerous Beetle
Emarald Tree Boa	Red-footed Booby	Brocket	Map Butterfly
Cacique	Dwarf Caiman	Black-headed Caique	Capuchin
Black-capped Capuchin	Capybara	Caracara	Crested Cariama
Cascabel	Andean Cat	Pampas Cat	Little Spotted Cat
Armoured Catfish	Blue Leopard Catfish	Pimelodid Catfish	Southern Mountain Cavy
Weasel Cavy	Cayman	Dragon Finned Characin	Rainbow Characin
Chinchilla	Long-tailed Chinchilla	Short-tailed Chinchilla	Cichlid
Dwarf Cichlid	Oscar Cichlid	Pike Cichlid	Triangle Cichlid
Greater Pichi Ciego	Lesser Pichi Ciego	Coati	Brown Coati
Ring-tailed Coati	False Cobra	Condor	Andean Condor
Boa Constrictor	Giant Cookroach	Giant Coot	Horned Coot
Flightless Cormorant	Land Crab	Rock Crab	Curassow
Black Curassow	Brazilian White-headed Curassow	Darter	Marsh Deer
Pampas Deer	Argentinian Pampas Deer	Swamp Deer	Bush Dog
Small-eared Dog	Amazonian Dolphin	Douroucouli	Inca Dove
Scaly Dove	Muscovy Duck	Torrent Duck	Black-chested Buzzard Eagle
Harpy Eagle	Ornate Hawk Eagle	Black and White Hawk Eagle	Electric Eel
Moray Eel	Plumbeous Forest Falcon	Spotted Silver Dollar Fish	Four-eyed Fish
Hatchet Fish	Hillstream Fish	Leaf Fish	Pampadour Fish
Pencil Fish	Andean Flamingo	James's Flamingo	Andes Fox
Crab-eating Fox	Southern Gray Fox	Greenhouse Frog	Horned Frog
Marsupial Frog	Arrow Poison Frog	Surinam Frog	Gallinule
Gar	Juan Fernandez Island Goat	Yellow Pocket Gopher	Grison
Brazialian Blue Grosbeak	Grunion	Guanaco	Guppy
Galápagos Hawk	Humpback Headstander	Little Blue Heron	Yellow-crowned Night Heron
Hoatzin	Hornero	North Andean Huemul	South Andean Huemul
Small Hummingbird	Hummingbird	Black-chinned Hummingbird	Black Faced Ibis
Scarlet Ibis	Iguana	Jaguar	Jaguarundi
Pileated Jay	Kinkajou	Everglades Kite	Cayenne Lapwing
Teiid Lizard	Llama	Peruvian Longfin	South American Lungfish
Mabuya	Macaw	Military Green Macaw	Hyacinth Macaw
Scarlet Macaw	Manatee	Amazonian Manatee	Caribbean Manatee
Mara	Margay	Buffy-headed Marmoset	Buffy-tufted-ear Marmoset
Common Marmoset	Cotton-top Marmoset	Black Eared Marmoset	Goeldi's Marmoset
Golden Lion Marmoset	Tassel-eared Marmoset	White Marmoset	Molly
Brown Howler Monkey	Red Howler Monkey	Black Spider Monkey	Brown-headed Spider Monkey
Geoffroy's Spider Monkey	Long-haired Spider Monkey	Woolly Spider Monkey	Squirrel Monkey
Red Titi Monkey	Woolly Monkey	Yellow-tailed Woolly Monkey	Morpho
Cypri's Morpho	Anopheles Mosquito	Motmot	Roufous Motmot
American Cane Mouse	Red Nosed Mouse	Ocelot	Oilbird
Wolly Opossum	American Oriole	Baltimore Oriole	Giant Brazilian Otter
Giant Otter	Marine Otter	La Plata Otter	Southern River Otter
Roufous Ovenbird	Burrowing Owl	Long Moustached Owl	Spectacled Owl
Paca	Pacarana	American Paddlefish	Pangolin
Parrakeet	Quaker Parrakeet	Yellow-fronted Amazon Parrot	Blue-headed Parrot
Red-fan Parrot	White-fronted Parrot	Chacoan Peccary	Collared Peccary
Brown Pelican	Galápagos Penguin	Humbolts Penguin	Rockhopper Penguin
Peropatus	Guinea Pig	Fruit Pigeon	Piranah
Thin-spined Porcupine	Tree Porcupine	Puculet	Nothern Pudu
Southern Pudu	Sooty Capped Puffbill	Puffbird	Puma
Forrest Rabbit	Crab-eating Raccoon	Clapper Rail	Giant Wood Rail
Chinchilla Rat	Spiny Rat	Rhea	Cock of the Rock
Saki	Southern Bearded Saki	Monk Saki	White-nosed Saki
Scalare	Screamer	Horned Screamer	South American Sea-lion
Elephant Seal	Juan Fernandez Fur Seal	Galápagos Fur Seal	Sloth
Maned Sloth	Brazilian three-toed Sloth	Blind Snake	Coral Snake
Canana Rat Snake	Urutu Snake	Pygmy Seed Snipe	Solendon
Atlantic Spadefish	Couch's Spadefoot	Bird-eating Spider	Silk Spider
Roseato Spoonbill	South American Red Squirrel	Variegated Squirrel	Jaribu Stork
Sunbittern	Black-necked Swan	Coscorba Swan	Bare-face Tamarin
Cotton-top Tamarin	Emperor Tamarin	Golden Lion Tamarin	Golden-headed Lion Tamarin
Golden-rumped Lion Tamarin	Red Mantled Tamarin	Pied Tamarin	Black and Red Tamarin
White-footed Tamarin	Tanager	Violet headed Tanager	Central American Tapir
Young South American Tapir	Mountain Tapir	Tarpon	Buenos Aires Tetra
Bentos Tetra	Cardinal Tetra	Bleeding Heart Tetra	Small Scaled Tetra
Tinamou	Masked Titi	Burrowing Toad	Columbian Toad
Darwin's Toad	Giant Tortoise	Galápagos Giant Tortoise	Yellow-legged Tortoise
Toucan	Red Billed Toucan	Saffron Toucan	Trogon
Violaceous Trogon	Ocellated Turkey	Black-headed Uakari	Red Uakari
Red and White Uakari	Vicuna	Viscacha	Black Widow
Maned Wolf	Woodcreaper	Black Cheeked Woodpecker	Bar Winged Wood Wren
Yapok	Zebu		

Table 14.1: Endangered South American Wildlife

Having located its own entry, a DSA will know its network address by looking at the `presentionAddress` attribute, hence it can start listening for operations.

But first it must load the database, this process is described in full in Section 14.8.

In its own entry it will find a `edbInfo` attribute, this describes which EDBs the DSA is expected to hold. The format of the `edbInfo` attribute is described fully in Section 14.10, but essentially the first parameter supplied says "load this EDB". Thus the attribute

```
edbInfo = ##
edbInfo = c=US@o=The Wollongong Group ##
edbInfo = c=GB ##
```

requests that the EDB files

quipu-db/EDB

(signified by no data before the first '#' sign),

quipu-db/c=US/o=The Wollongong Group/EDB

and

quipu-db/c=GB/EDB

are loaded.

There are a few other important attributes your DSAs entry should have, the "toucan" entry in Figure 14.2 gives examples of the other attributes. They are briefly described below.

description: A Textual message describing the DSA, and the wildlife!

quipuVersion: This should contain the version number of the QUIPU software you are using. This can be found by using the dish command `squid -version`. You only have to set this once. From then on the DSA will manage it, updating the entry when required.

manager: The DN of the manager of the DSA, this user will not be blocked by access control when modifying the local database over DAP.

supportedApplicationContext: This should always have the value

```
QuipuDSP & X500DSP & X500DAP
```

for this version of QUIPU. It is used by remote DSAs to decide which protocols your DSA supports, and thus how best to contact it.

objectclass: This must contain "`quipuDSA`".

listenAddress: Sometimes (particularly in the X.25 world, or when using a transport service bridge) the address a server should listen on, is not the same as the address clients should call. The presentationAddress attribute defines the address clients will always call. If, and only if, this is not the address the server should listen on, the address for the server should be given in the listernAddress attribute.

acl: A DSAs entry, like any other, can be protected by access control lists. Care should be taken to make sure all remote DSAs can see all the attributes they need. The following ACL is recommended for use in a DSAs entry. The string "`MANAGER DN`" should be replaced with the distinguished name of the DSA manager as defined by the "manager" attribute.

```
acl= group # MANAGER DN # write # entry
acl= group # MANAGER DN # write # default
acl= others # read # entry
acl= others # read # default
acl= others # read # child
```

A little explaination may help as this might seem somewhat cryptic. The "`others # read # child`" line is to prevent anybody from adding children to the entry, it is the only line referencing the children, so nobody is given write access. The userpassword attribute in the entry does not need protecting for two reasons. Firstly nothing can make use of it, "self" has only got read access to the entry (as defined by the others clause). Secondly, the password will probably be in a highly replicated EDB file, so despite the pseudo crypting, is easily breakable.

relayDSA: If your DSA is not connected to one of the major networks (Internet, IPSS...), it may from time to time get references to a DSA that it can not connect to directly. For the operation to succeed, your DSA will need to chain the operation to a DSA that can progress the query.

This attribute is the DN of a DSA or DSAs that are connected to both your network and the major networks you are not connected to. There needs to be an agreement between the managers of the two DSAs because the relay DSA will be asked to perform operations on your behalf. For example, if the DSA "x" has access to IPSS and Internet, but DSA "y" only has IPSS access, DSA "y" might add a relayDSA attribute to it own entry, with the DN of DSA "x" as the value. Then, when DSA "y" gets a reference to an Internet based DSA, it will chain the operation the DSA "x".

Clearly, if every DSA chooses the same relay DSA, that DSA will soon become overloaded and reject your association attempts with a "busy" error. So some care is needed in choosing the "right" DSA (The QUIPU team recognise that there needs to be some form of "relay authorisation" and are looking at possible solutions for future versions of QUIPU). Section 14.3 describes how to prevent your DSA (e.g., DSA "x") from chaining operations on behalf of other DSAs.

photo: A picture of the wildlife !

Section 14.7 describes how you can add data to your DSA by extending the supplied textual database to include your own data or by sending data to the Directory via the DUA modify operations. This may be done independently from connecting to the global directory.

14.3 Tailoring

The previous sections have described how to start a basic DSA, and connect a DUA to it. Having done this, there are various configuration options you can set, which are described in this section.

On start up the DSA first consults a run time tailor file (the default DSA tailor file is called *quiputailor* in the ISODE `ETCDIR` directory, but can be changed with a `-t` parameter to *ros.quipu*; consult *quipu*(8c) for details), which indicates such things as:

- name of the DSA
- location of the database
- location and level of the logs that the DSA will produce.
- location of any other DSAs for initial bootstrap

At startup, the *isotailor*(5n) file is also consulted to configure the system wide ISODE parameters.

The format is identical to the DUA tailor file described in Section 13.2, with the addition of the following options:

`mydsaname:` The distinguished name of the DSA. For example,

```
mydsaname   cn=Axolotl
```

declares this DSA to have the common name of `Axolotl`. Quotes will be needed in the name contains a space characer, for example

```
mydsaname   "c=GB@cn=Long Moustached Owl"
```

`parent:` This entry consists of a name/address pair of a parent DSA. The DSA referenced needs to hold a Master or Slave copy of an EDB higher up in the DIT than the highest locally held EDB. For example,

```
parent C=GB@CN=vicuna Internet=vs1.ucl.cs.ac.uk+50987
```

declares the parent DSA to be "C=GB@CN=vicuna" at the indicated address. If more than one `parent` tailor entries are found, the DSA will chose which DSA to contact. The first DSA in the list is taken as the "master" reference, with the subsequent entries as "slave" references.

If your DSA holds a copy of the ROOT EDB file, this parameter should have just one value which is a reference to the DSA holding the MASTER copy of the ROOT EDB file (currently "cn=Giant Tortoise").

`stats`: The value has the same format as the `dsaplog` entry described in Section 13.2, and is used to control the level of DSA statistical logging.

`treedir`: Defines the directory in which the textual database is stored. For example,

```
treedir     /usr/etc/quipu-db/
```

declares the directory */usr/etc/quipu-db/* to contain the local part of the Directory Information Tree. Be sure to remember the trailing slash "`/`".

`shadow`: When searching, often a large part of the time is involved with chaining off to other DSAs to search aliases. To enhance performance it is sometimes useful to have a cached copy of the alias locally, this tailor variable allows such attributes to be "spot shadowed". The value of this variable is an attribute type. The attribute value associated with this type should have DN syntax. When the database has been loaded, if any instance of this attribute references a DN that is not held locally your DSA will "spot shadow" that entry. That is from time to time it will read the entry from a remote DSA, and cache the result.

`optimize_attr`: If TURBO_AVL and TURBO_INDEX have been defined, this option specifies the attribute types to index. Each attribute type must be on a separate line. For example the lines:

```
optimize_attr commonName
optimize_attr surname
```

would arrange to index both the commonName and surname attributes. Only string attributes are allowed. NOTE: this option must come before any `index_subtree` or `index_siblings` options.

`index_subtree`: If TURBO_AVL and TURBO_INDEX have been defined, this option specifies the distinguished name of a subtree to index for subtree searches. Multiple subtrees may be specified by multiple lines. For example, the lines

```
index_subtree "c=US@o=your org"
index_subtree "c=US@o=your org@ou=really big OU"
```

arrange to have two indexes built, one for each subtree specified. Be aware that the indexes take up extra space in core, so care should be taken to index things sparingly.

`index_siblings`: If TURBO_AVL and TURBO_INDEX have been defined, this option specifies the distinguished name of the parent of a group of siblings to index. For example, the line

```
index_siblings "c=US@o=your org@ou=really big OU"
```

arranges to have an index built for one-level searches directly below the specified entry. Multiple indexes may be specified by multiple lines. Be aware that the indexes take up extra space in core, so care should be taken to index things sparingly.

`optimized_only`: If TURBO_AVL and TURBO_INDEX have been defined, the value "on" tells the DSA to refuse any searches that do not consist entirely of optimized attributes and filters. Any such non-indexed queries will be rejected with an "unwilling to perform" service error.

`update`: The value "on" tells the DSA to update SLAVE and CACHED EDB files when it starts up. See Section 14.10 for further details, by default this parameter is "off".

`searchlevel`: Defines the level below which users will be able to search the DIT — default 2 (e.g., below organisations). If they try to search from higher up, a "unwilling to perform" service error will result.

`lastmodified`: If the value is "`off`", the attributes "`lastmodifiedby`" and "`lastmodifiedtime`" will not be added by the DSA when an entry is altered.

`readonly`: Bring the DSA up in "read only" mode — that is prevent user modification. Slave updates are still allowed.

`dspchaining`: The value "on" tells the DSA it is allowed to chain DSP requests to other DSAs if necessary. The default mode of operation is to return a DSA referral, unless a chain is need due to a disjoint network. The full set of issues deciding whether to use chaining or referrals is discussed in the QUIPU design document ([SKill89b]), and [PBark89].

`adminsize`: The administrative size limit for use on search and list operations.

`admintime`: The maximum time allow to spend on a user query.

`cachetime`: The length of time to keep / use "cached" information.

`conntime`: The length of time to hold a unused connection open.

`nsaptime`: The length of time to wait before deciding a connection can not be established for a given NSAP.

`retrytime`: The length of time before deciding it is worth attempting to connect to a DSA that could not be contacted earlier.

`slavetime`: The length of time between attempting to update slave EDB files.

`preferdsa`: When a DSA has creates a reference to other DSAs it tries to discriminate between and chose the "best" DSA to contact. This variable gives you a handle on controlling the choice. For example the lines

```
preferDSA c=GB@cn=Vicuna
preferDSA "cn=Giant Tortoise"
```

tells the DSA to use either "c=GB@cn=Vicuna" or "cn=Giant Tortoise" in preference to any other DSA. Further, use the DSA "c=GB@cn=Vicuna" rather than "cn=Giant Tortoise" if possible.

`cainfo`: For authentication.

`secretkey`: For authentication.

`bindwindow`: For authentication.

`isode`: The argument is an `isodevariable` `isodevalue` pair as would be found in *isotailor*. This is used to "override" isotailor settings.

14.3.1 Tailoring a Running DSA

The tailoring described above is performed when the DSA is booted. It is sometimes useful to alter the tailor settings after the DSA has started without having to bring the DSA down and rebooting. This can be done by via the special "**dsacontrol**" command in the *dish*(1c) program as described in Section 4.5 on page 44.

The DUA invokes a modify operation, with a single special attribute "dSAControl" defined in [SKill89b]. The DSA recognises the special attribute, and provided you are bound as the manager of the DSA, passes the attribute value to the appropriate routine.

[SKill89b] describes this process in full.

14.4 Modifying your DSAs Entry

A DSA manages its own entry in the DIT. Generally the MASTER EDB in which your DSAs entry resides is not held by your DSA. For security reasons, this means it is difficult for a DSA to modify its own entry directly, so for example it can't keep its version number uptodate. However, a DSA holding the MASTER EDB knows that any QUIPU DSA[4] wants to manage its own entry. To allow this to happen, the DSA holding the MASTER EDB "spot shadows" the remote DSA entry. That is, from time to time it connects to the DSA in question, reads its entry, and writes the result back into the MASTER EDB file. So modify operations on the DSAs entry can now take place in your local DSA. This has the advantage that attributes such as the version number are kept up to date.

To perform this shadowing, the DSA has to read its own entry across an un-authenticated DSP link, thus it can not read any attributes that are

[4] Any QUIPU DSA for which the version number is 6.8 or greater

protected by ACLs. So all important attributes in the DSAs entry MUST be publically readable (this includes the unused userPassword attribute). If they are not readable the shadowing operation will fail.

To keep the database consistent with the MASTER EDB, when a modify takes place, your DSA can not re-write the information back to its SLAVE copy EDB. So it writes the entry in a file called *DSA.real* in the top level of your database. If there is an inconsistency found, it is this file that is "trusted".

There is also a file called *DSA.pseudo* which contains some attributes managed by the DSA, that it does not need to make public (such as which cached EDB files it holds). You should NEVER need to edit this file.

There is a problem when modifying the presentation address of a DSA. You must make sure the DSA with the MASTER EDB reads the new address, BEFORE you move the DSA. If not it will always attempt to connect to the wrong address to try to shadow the entry, and never find the new address (Alternatively you could use a ts_bridge to make it looks as if the old address still works). You can check the MASTER EDB has updated by making a DAP bind to the DSA managing the EDB and reading the address it has got. If it is out of date, you should wait for it to be updated, this usually takes place every six hours. To summarise, the sequence of events is

1. Change the address attribute using a DAP modify operation,
2. Wait for the DSA with the MASTER EDB to read the entry,
3. Restart the DSA on the new address.

14.5 Connection to Other DSAs

All QUIPU DSAs should be connected. This section describes how you should configure your DSA to connect to other DSAs in order to read the data not held locally. Section 14.5.1 describes how to make sure that other DSAs can access your DSA.

A dynamic approach is used to bootstrap a new DSA. The "global master" DSA is administered at the University of London Computer Center[5] (ULCC), and other DSAs should be configured in relation to this one.

[5] The DSA has moved from University College London

Every DSA needs to know the distinguished name and presentation address of one or more DSAs nearer to, or actually holding, the root EDB. This parameter is supplied in the tailor file (see Section 14.3) under the name `parent`. This information is used by the QUIPU DSA when it can not find a DSAs entry in the local database, or can find no pointers in the local database to the whereabouts of the data. If you hold a `SLAVE` copy of the root EDB, then your parent should be a DSA that is "closer" to the MASTER copy of the root EDB, and probably the MASTER itself.

The example databases are set up with default parents (defined in the file *quipu-db/organisation/quiputailor*). To see if your DSA can contact the "parent", connect to the local test (root holding) DSA using `dish`, and type

```
list @
```

This will list the locally held root, now try

```
list @ -dontusecopy
```

This will try to connect to the parent DSA.

Care should be taken in choosing the parent DSA. If all DSAs have the same parent DSA then that DSA may become overloaded. Typically each site will have several DSAs. One of these should be the default parent for all the other DSAs, with only one DSA having a default parent outside the site.

When "walking down" the DIT, QUIPU needs access to information to tell it on which DSA the next level of the DIT is stored. To do this each non-leaf entry MUST have a "masterDSA" or "slaveDSA" attribute. The value of the attribute is the distinguished name of the DSA holding the next level of the DIT (it may be your own DSA name, if you hold the next level as well). If there is no such attribute, QUIPU assumes the entry is a leaf. If the information required is not held in the local DSA then QUIPU chains the request to the named DSA or returns a referral — depending upon the mode of operation (The named DSAs entry is read to establish the address — this may mean a connection to another DSA in-order to read the entry).

Once you have started your DSA, you should try to connect to other DSAs using the `DISH` program (this connects to the local DSA, which will chain requests to other DSAs). Try listing the children of organisations other than yourself, to find which organisations are connected, try listing your countries children.

14.5.1 Connection to the Global Directory

To enable the global directory to connect to you, you must contact the manager of the DSA immediately above the node you want to added under. Supply them with the entry for your DSA, and the top level entry of the sub-tree you want to hold. For example, to add the organisation "o=foobar" below "c=US", contact the manager of the DSA holding "c=US" as a MASTER, giving them the entry for "c=US@o=foobar" and "c=US@cn=foobar_dsa", assuming "c=US@cn=foobar_dsa" is the name of your DSA (see Section 14.2.2 on naming a DSA).

By convention, to find out who administers the master EDB for a particular node, run the *dish*(1c) program and retrieve the `manager` attribute from the entry for the DSA holding that node.

You can then find a mail address by looking that person up in the directory.

The following example shows you how to get the mail address of the manager of the c=GB subtree.

```
% dish
Welcome to Dish (DIrectory SHell)
Dish -> showentry @c=GB -type masterDSA
masterDSA  - cn = Giant Tortoise
Dish -> showentry "@cn=Giant Tortoise" -type manager -edb
manager= c=GB@o=X-Tel Services Ltd@cn=Camayoc
manager= c=GB@o=University College London@ou=Computer Science@cn=incads
Dish -> moveto "@c=GB@o=X-Tel Services Ltd@cn=Camayoc"
Dish -> showentry -edb
cn= Camayoc
roleOccupant= c=GB@o=X-Tel Services Ltd@cn=Colin Robbins
objectClass= organizationalRole & quipuObject
Dish -> moveto "@c=GB@o=X-Tel Services Ltd@cn=Colin Robbins"
Dish -> showentry -type rfc822mailBox
rfc822Mailbox         - C.Robbins@xtel.co.uk
Dish -> quit
%
```

If you want to add data to the ROOT node of the "global tree". The new entries should be sent to *QUIPU-support* at the address given in Section 2.6.

When you think you are connected to the "global DIT", you should test that you are. To do this use `dish` to connect to a DSA higher than you in the DIT (using the `-call` flag as described in Section 13.2, then see if you can navigate to your own part of the DIT, and look at your data — if so then the DSA is connected.

14.6 Connecting to a Non-QUIPU DSA

QUIPU has the concept of mastering all entries in one EDB file built into its design. Other implementations may not share this model. The entry represented by the non-QUIPU DSA can be "spot shadowed". For example, if you wanted to add "o=foobar" to the DIT and this was mastered by a non-QUIPU DSA, you would add an entry of the form

```
o=foobar
objectClass= ExternalNonLeafObject & organization
subordinateReference= cn=The DSA # Internet=123.456.1.2+17003
```

where "cn=The DSA" is the DN of the DSA, and "Internet=..." is the presentation address of the DSA. Instead of a subordinate reference, a cross reference or non-specific subordinate reference can be given using the "crossReference" or "nonSpecificSubordinateReference" attributes respectively.

Although this is entry is in a MASTER EDB file, QUIPU recognises that is may not actually hold the authorative master, and contacts the referenced DSA as required.

14.7 Adding more Data

Having got a DSA started, and connected to the global DIT, you should start to add lots of data. There are many sources of such data, and with just a relatively small amount of effort this data can be added to the directory.

There are two ways such data can be added. First of all you can use one of the DUA programs to bind to the DSA as the DSA manager, and send data to the directory via the "add" and "modify" operations (see Sections 4.1.5 and 4.1.8). This is probably the best way to add relatively small amounts of data, or make minor changes to the data.

To add a large amount of data (i.e., the initial creation of a large database) if is probably easiest to write a shell script using UNIX tools such as `awk` and `grep` to create the EDB files directly. In the next version of the software there will be tools to facilitate this.

When adding data for users it is advisable to allocate a "userPassword", and a suitable "ACL" to protect this password. Chapter 11 describes how to do this.

14.7.1 More on Object Classes

The only object class discussed so far is the "quipuDSA" objectclass. When you start to add data, you will probably want to add information about people, sub-divisions of your organisation, and other application entities. This section introduces some of the more important object classes, and the attributes they may contain. In many cases, only the attribute type is specified, for details of typical values and the value syntax you should read Chapter 10.

As already described, every entry in the DIT must belong to the object class `top`, which means every entry must have an `objectClass` attribute. Also, every entry in the QUIPU DIT should belong to either the `quipuNonLeafObject` or `quipuObject`.

The following part of this chapter describes some of the basic objectclasses and the attributes implied.

Person

This is a base object class used to represent a person.

There are two mandatory attributes:

commonName: which gives a (potentially ambiguous) name for the person. The value of this attribute is a string usually containing the person's first and last names; e.g.,

```
Marshall Rose
```

This attribute is usually multi-valued, containing variations on the first, middle, and last names; e.g.,

```
Colin Robbins
Colin John Robbins
Colin J. Robbins
```

Generally this attribute will supply the distinguished attribute of the entry.

surName: which gives the person's last name.

The optional attributes are:-

`userPassword`	`seeAlso`
`telephoneNumber`	`description`

OrganizationalPerson

This is a sub-class of the `person` person object class and introduces the following optional attributes:-

`preferredDeliveryMethod`	`destinationIndicator`
`registeredAddress`	`internationaliSDNNumber`
`x121Address`	`facsimileTelephoneNumber`
`teletexTerminalIdentifier`	`telexNumber`
`physicalDeliveryOfficeName`	`postOfficeBox`
`postalCode`	`postalAddress`
`title`	`organizationalUnitName`
`streetAddress`	`stateOrProvinceName`
`locality`	

ThornPerson

like `OrganizationalPerson`, this is also a sub-class of `person` and introduces the following optional attributes:-

`homePostalAddress`	`lastModifiedBy`
`lastModifiedTime`	`secretary`
`homePhone`	`userClass`
`photo`	`roomNumber`
`favouriteDrink`	`info`
`rfc822Mailbox`	`textEncodedORaddress`
`userid`	

Two example `ThornPerson` entries are given in Figure 14.1 on page 124.

OrganizationalRole

Entries of this class are used to represent a position or role within an organization.

There is one mandatory attribute:

commonName: which gives the name of the role. The value of this attribute is a string; e.g.,

```
PostMaster
```

There are many optional attributes including:

roleOccupant: which is the Distinguished Name of the person who fulfills the role e.g.,

```
c=US@o=X-Tel Services Ltd@cn=Peter Cowen
```

The other optional attributes are:

```
seeAlso                        preferredDeliveryMethod
destinationIndicator           registeredAddress
internationaliSDNNumber        x121Address
facsimileTelephoneNumber       teletexTerminalIdentifier
telexNumber                    telephoneNumber
locality                       postOfficeBox
postalCode                     postalAddress
description                    organizationalUnitName
streetAddress                  stateOrProvinceName
physicalDeliveryOfficeName
```

Alias

Objects of this class represent an alias to some other entry in the DIT. It is generally used when an entity belongs in one or more subtrees of the DIT, and is used to "point" one entry at the other.

There are two mandatory attributes:

commonName: which gives the name of the alias. The value of this attribute is a string; e.g.,

```
Colin Robbins
```

aliasedObjectName: which is a pointer to another object in the Directory; e.g.,

```
c=GB@o=X-Tel Services Ltd@cn=Colin Robbins
```

There are no optional attributes for this object class.

An example of an Alias entry is given below:-

```
cn= QUIPU-support
aliasedObjectName= c=GB@o=University College London
   @ou=Computer Science@cn=Incads
objectClass= quipuObject & alias & top
```

OrganizationalUnit

The `OrganisationalUnit` object class is used to represent a unit within your organisation. There is one mandatory attribute

organizationalUnitName: which gives the name of the organizational unit. The value of this attribute is a string; e.g.,

```
Research and Development
```

The optional attributes are:-

`userPassword`	`seeAlso`
`preferredDeliveryMethod`	`destinationIndicator`
`registeredAddress`	`internationaliSDNNumber`
`x121Address`	`facsimileTelephoneNumber`
`teletexTerminalIdentifier`	`telexNumber`
`telephoneNumber`	`physicalDeliveryOfficeName`
`postOfficeBox`	`postalCode`
`postalAddress`	`businessCategory`
`searchGuide`	`description`
`streetAddress`	`stateOrProvinceName`
`locality`	

Organization

Objects of this class represent a top-level organizational entity, such as a corporation, university, government entity, and so on.

There is one mandatory attribute:

organizationName: which gives the name of the organization. The value of this attribute is a string; e.g.,

```
        Performance Systems International
```

The optional attributes are:-

```
userPassword                    seeAlso
preferredDeliveryMethod         destinationIndicator
registeredAddress               internationaliSDNNumber
x121Address                     facsimileTelephoneNumber
teletexTerminalIdentifier       telexNumber
telephoneNumber                 physicalDeliveryOfficeName
postOfficeBox                   postalCode
postalAddress                   businessCategory
searchGuide                     description
streetAddress                   stateOrProvinceName
locality
```

domainRelatedObject

If an object has some relationship to the Internet Domain Name System (DNS), then this can be represented in the DIT using this objectclass.

This class has one mandatory attribute:

associatedDomain: identifies the domain which corresponds to this object. The value is a domain string, e.g.,

```
psi.com
xtel.co.uk
```

14.7.2 Schemas

Directories should provide a very flexible tool which enables any information to be stored. There is a danger that Schemas, as specified in the OSI Directory, will lead to procrustean directory implementations which impose unreasonable restrictions. The QUIPU Directory does not, per se, place restrictions on what can be placed in a DSA. It does however give control so that managers may control what is stored in the directory.

The first aspect of structure is with respect to attributes which may be present in an entry. A QUIPU DSA will allow an entry to belong to one or more object classes which are known to the DSA (stored in a table). An entry will typically have a small number of object classes (e.g., TOP (implicit) + Person + Organisational Person + QUIPU Object). The DSA will maintain a table of mandatory and optional attributes for each object class supported. This will follow the guidelines of the standard or specification identifying the object class in question. From this information, the DSA can determine the permitted and mandatory attributes for a given entry, by calculating the union of all the object classes of that entry. Free extension (i.e., the ability to store any attribute) was rejected, as there does not appear to be a reasonable mechanism to manage this. However, it is straightforward for managers to create new object classes as desired.

```
TreeStructureSyntax ::= SET {
    mandatoryObjectClasses [1] SET OF OBJECT IDENTIFIER,
    optionalObjectClasses [2] SET OF OBJECT IDENTIFIER OPTIONAL,
    permittedRDNs [3] SET OF SET OF AttributeType }

TreeStructure ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX TreeStructureSyntax
    MULTI VALUE
```

Figure 14.3: Schema definition

It is important to allow management control of what is permitted at a given level. Therefore a "tree structure" attribute may be created. This attribute is defined in Figure 14.3, with the string syntax discussed in 10.2.2. This specifies for the level below, what types of object are permitted. Each attribute value identifies a class of object which can exist at the level below, and defines a set of mandatory and optional object classes. This can be considered as defining a (private) object class implied by the combination of these classes.

For each type of object, the attribute types permitted in the RDN are also listed. This is not checked in the current version of QUIPU. The directory knows about the tree structure attribute, and will ensure consistency. When creating an entry, the DSA must check that it conforms to the treeStructure attribute of the parent entry. When removing information from a treeStructure attribute, the Directory will check that all of the children conform to the modified attribute.

14.7.3 Photograph Attributes

The data that a DSA can hold does not have to be limited to data that can be represented by printable strings. Any attribute a QUIPU DSA does not know a string syntax for, it will hold as a block of ASN.1. One such attribute is the "photo" attribute. Photographs of people and objects are stored in the directory as a g3fax encoded block of ASN.1, and are best stored as "file" attributes described in the next section. There is an example of a photograph attribute in the example database. This can be looked at by connecting to the directory with `dish` and looking at the entry "Steve Kille, CS, UCL, GB"

Unless you you have compiled and installed the "photo" code as described in Section 12 you will see the message

```
''No display process defined''
```

Having compiled the "photo" code, you will need to add a line such as

```
photo xterm Xphoto
```

to your dsaptailor file (see Section 13.2) This says that if the user in logged on to a terminal of the type `xterm` then use the process *g3fax/Xphoto* in the ISODE `ETCDIR` directory to display the photograph.

There are display routines provided for the X Window System, SunView,Tektronix 4014, and "dumb" terminals.

Generating Photos

The photograph files used by QUIPU need to be two dimensionally G3fax encoded. QUIPU contains some tools to help you get your files into the requied format. There are found in the *others/quipu/photo* directory of the ISODE source tree.

Getting pictures onto your machine is a local problem, that will need discussing with your system manager. You may have them in a different format to that required by QUIPU.

If they are in SunView pixrect format, the tool "pr2pe" can be used to convert them. For other formats, there are two utilities "pbmtofax" and "faxtopbm", which can convert from the "PBM" format to two dimension fax, and vice verca. The PBM format is that defined and used by Jef Poskanzer's pbmplus package. This package can convert a large range of bitmap formats into the PBM format[6]. You will need to obtain this package to compile the "pbmtofax" and "faxtopbm" utilities.

NOTE with this version of QUIPU, the "pbmtofax" tool should always be used with a "-old" flag to ensure user at remote sites with older code can still decode and display your photograph files.

14.7.4 File Attributes

Attributes values are generally stored in the EDB file, and loaded into memory when the DSA starts. For some large attributes such a photos, this is not a sensible approach, so the concept of a "file" attribute is introduced.

If an attribute value is prefixed by `FILE`, then the value is assumed to be stored on disk. For example if the following were found in an EDB file:-

```
photo={FILE}/usr/local/pictures/steve
```

then the value for the photo attribute would be read from the file

`/usr/local/pictures/steve`.

The syntax of the data in the file is expected to be the same as the string syntax, except in the case of ASN.1 and PHOTO attributes which are expected to be in raw ASN.1.

If there is not a file name supplied, then a default name is allocated. The default file name is the RDN of the entry the attribute belongs to followed by a dot ".", followed by the attributeType. This file is expected to be in the same directory as the EDB file for the entry. For example the default file for photo attribute, representing the entry for "c= GB @ o= University College London @ ou= Computer Science @ cn= Colin Robbins" would be:-

```
cn=Colin Robbins.photo
```

in the directory

```
quipu-db/c=GB/o=University College London/ou=Computer Science
```

[6] You should not confuse "pbmtofax" with the "pbmtog3" tools which is part of the pbmplus package — this uses a one dimensional encoding scheme

or equivalent mapped file as described in Section 14.1.4.

The process defined so far allows for attribute stored in a EDB in file format to be read by a DSA. Writing the attributes back to files if modified/added by a DUA requires a little more. The DSA needs to know which attributes should be stored on disk. This information is supplied in the attribute oidtables which are defined in Section 14.11. For example if the following entry was found in the file *oidtable.at* in the ISODE `ETCDIR` directory then a "photo" attributed would always be written back to disk.

```
photo:  thornAttribute.7: photo : file
```

14.7.5 Attribute Inheritance

Attribute inheritance is a mechanism where an entry can get default attribute values from its parent entry. This can make management of common attributes much easier. Inheritance can be used to make the in core database significantly smaller, which as a side effect should make the DSA run faster. For example, entries of the object class "person" for a particular organisation might all have the same postal address attribute. Using inheritance this attribute can be placed in the organisation entry and inherited down to all "person" entries.

The information is placed in the entry above using the "inheritedAttribute" attribute type, defined in Figure 14.4. The QUIPU string syntax to represent this is:

```
<InheritedAttributeValue> ::= [ <oid> "$" ]
                        [ "always" <InheritedList> ]
                        [ "default" <InheritedList> ]
<InheritedList> ::= "(" NEWLINE <AttributeSequence>
                        NEWLINE ")"
```

For example, if the following was an attribute of an entry

```
inheritedAttribute = person $ always (
postalAddress= UCL $ Gower Street $ London $ WC1E 6BT
postalCode= WC1E 6BT
) default (
telephoneNumber= +44 71-387-7050
)
```

```
InheritedListSyntax ::= SET OF CHOICE {
   AttributeType,          -- Take value from the entry
   Attribute }

InheritedAttributeSyntax ::= SET {
   default [0] InheritedListSyntax OPTIONAL,
      -- default which can be overriden in lower entry.
   always  [1] InheritedListSyntax OPTIONAL,
      -- always present in lower entry.
   objectclass OBJECT IDENTIFIER OPTIONAL
      -- object class to inherit to.
      -- The null case means the objectclass attribute
      --    itself is inherited
}

InheritedAttribute ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX InheritedAttributeSyntax
   MULTI VALUE
```

Figure 14.4: Attribute Inheritance

then all entries of objectclass “person” positioned one level below the entry would always inherit the attributes

```
postalAddress= UCL $ Gower Street $ London $ WC1E 6BT
postalCode= WC1E 6BT
```

and **if** the entry does not contain a telephoneNumber attribute itself, it would inherit

```
telephoneNumber= +44 71-387-7050
```

The attribute value in the inherited attribute can be left blank, for example

```
inheritedAttribute = person $ always (
postalAddress
)
```

in which case the value is taken from the same entry as the inherited attribute itself. This avoids duplicating attributes.

The OID representing the objectclass can only appear in one attribute value, thus the following, which may seem similar to the above is **NOT** allowed, they need to be combined.

```
inheritedAttribute = person $ always (
postalAddress= UCL $ Gower Street $ London $ WC1E 6BT
)
inheritedAttribute = person $ always (
postalCode= WC1E 6BT
)
inheritedAttribute = person $ default (
telephoneNumber= +44 71-387-7050
)
```

If the OID representing the objectclass is omitted, then the values are inherited to entries that themselves do not contain an objectclass attribute, thus the inherited attribute must define an objectclass to inherit down (as schema checking still takes place). For example

```
inheritedAttribute = default (
ObjectClass= person & quipuobject
postalAddress= UCL $ Gower Street $ London $ WC1E 6BT
postalCode= WC1E 6BT
)
```

Inheritance can be used for any attribute with the following exceptions:

- An entry can not inherit its distinguished attributes.
- An inherited userPassword attribute in not used to authenticate a user at bind time.
- The following system attributes

objectClass	ACL
masterDSA	slaveDSA
aliasedObjectName	edbInfo
presentationAddress	relayDSA
treeStructure	supportedApplicationContext
inheritedAttribute	

can only be used in the "default" clause. They can not appear in the "always" clause as there is the potential for an inconsistency to occur.

- You must hold at least a copy of the EDB file containing the parent entry.

If an entry belongs to more than one object class, and more than one of these classes is listed in the inherited attribute of the parent entry, then only attributes from ONE of these will be inherited into the entry. If is undefined which.

The inheritance described thus far only covers one level of the DIT, whereas it is often useful to inherit attributes down the whole subtree. This can be achieved by inheriting the inheritance attribute! Consider the multi-valued attribute

```
iattr= organizationalUnit $ default (
iattr
)
iattr= person $ default (
telephoneNumber
)
```

Every "person" immediately below will inherit the telephone number attribute. Every "organizationalUnit" will inherit the same inheritance attribute, thus any "person" entries below "organizationalUnit" entries will get the telephone number attribute as well. by the same token, this works for multiple levels of organizationalUnit.

14.8 How a DSA Starts

The section gives a brief description of how a QUIPU DSA starts. This is intended to help people who have experienced problems in getting their DSAs started.

First of all the tailor file *quiputailor* in the ISODE `ETCDIR` directory is read (unless you specify a different tailor file using the `-t` option to *ros.quipu*). This tells the DSA, amongst other things, its own name, and its parent DSA(s).

The first thing a DSA need to do if find its own entry, thus it tries to load the EDB that should contain it. If it can not find its own entry, then it will try connecting to the parent DSA to read its own entry. If this fails the DSA will stop.

Having found its own entry, the DSA will check to see if it is a cached entry read fron disk, if so it will attempt to read a new version from the relevant remote DSA — if successful a new cache EDB file will be written.

Now the rest of the local DIT can be loaded. Each EDB specified in the "edbInfo" attribute is in turn loaded from disk, followed by any cached EDB files.

As each EDB is loaded from disk, it is checked to make sure all the attributes in the entry are allowed, as defined by the "objectClass" attribute, and that the tree shape conforms to that defined by the "treeStruture" attribute. If any EDB fails these checks then the DSA will not start, and an appropriate message will be logged.

Having loaded all its data, the DSA will start listening for DAP and DSP associations.

14.9 Adding more DSAs

Many organisations will need to have more than one DSA to meet their requirements, the section suggests how you might arrange your DSAs to get the most out of the system.

You will almost certainly need at least one DSA that holds a copy of the root EDB, and your country EDB. This DSA should also hold the MASTER copy of your organisations own EDB. Subsequent DSAs that are set up, should have this DSA defined as one of their parent DSAs.

For robustness, it is a good idea to have some other DSAs replicating these EDBs, so that if one local machine or DSA becomes unavailable, then there is another copy elsewhere that can be used. EDBs that contain addresses of other DSAs that you may want to contact regularly should also be replicated, to prevent extra associations having to be made just to read DSA addresses. In short, if you know that data from a certain EDB is going to be accessed a lot, replicate it locally. Replication does not have a high cost.

When setting up multiple DSAs be sure to name them as described in Section 14.2.2, ensure the DSA entries are sufficiently high in the DIT, so that other DSAs can read the entry, without having to contact the DSA concerned.

14.10 Receiving EDB Updates

If your DSA holds a slave copy of one or more EDBs, then it can automatically update these for you. The current approach is very simple minded, but will be extended for future versions.

There are two ways to ask a DSA to update its slave EDBs, either use *dish* program and issue a "dsacontrol -slave" command — see Section 4.5, or set the `quiputailor` parameter "update" to "on" (see Section 14.3) — in which case the DSA will try to update its slaves when it starts up.

To update the slaves, the DSA uses the "edbinfo" attribute that the DSAs entry in the DIT MUST have. This attribute specifies which EDBs you hold, and where to get updates of the from. NOTE you do not necessarily have to get updates from a `MASTER` EDB, a `SLAVE` is acceptable. In many cases this will be preferable to prevent the load on the `MASTER` DSA being too high. For example a "national" EDB is likely to be highly replicated, it would not be a good idea to have just one DSA handling updates. It would be better to have the load spread over several DSAs.

The syntax of the attribute is

```
edbinfo = EDB concerned # get from # send to
```

It is a multi-valued attribute.

A few examples:

```
# cn=Giant Tortoise # cn=Fruit Bat
c=US # # cn=Fruit Bat
c=US # # c=US@cn=Spectacled Bear
c=US@l=NY # cn=Fruit Bat #
```

Note that there is no harm to using multiple `eDBinfo` lines, even if they refer to the same EDB. These lines indicate that:

- The ROOT EDB is read from the `cn=Giant Tortoise` DSA, further the `cn=Fruit Bat` DSA is allowed to read the ROOT EDB from this DSA. This is an important point — A DSA has to explicitly say it will allow you to update from it, before it will send EDB files. Thus if you want to pull EDB files from a remote DSA, you will need to ask the manager to add you DSA to their DSAs "send to" field.

- The `cn=Fruit Bat` DSA and the `c=US@cn=Spectacled Bear` DSA are allowed to read the EDB for `c=US`. Note that the second and third line could be combined as:

  ```
  c=US # # cn=Fruit Bat$c=US@cn=Spectacled Bear
  ```

- The DSA named `cn=Fruit Bat` supplies the EDB for `c=US@l=NY` to this DSA.

Some EDB files, such as the ROOT, and country level EDB files are highly replicated, and having to list all the DSAs that are allowed to pull such EDBs is too restrictive. Thus the following syntax can be used as an abbreviation:

```
c=US # # c=US
```

This indicates that the c=US EDB file can be sent to DSA named at the c=US level of the DIT, for example

```
c=US@cn=Wombat
```

is allowed to pull the EDB, but the following are not:

```
cn=Wombat
c=US@o=Foobar Inc.@cn=Wombat
c=GB@cn=Wombat
```

Whether to update is decided upon by looking at the version string, if the two EDB files have a different version string, then the EDB will be updated.

14.11 Tables

Throughout QUIPU, all the Object Identifiers that need to be specified, are specified using a string representation, for example:-

String representation	Object Identifier
attributeType	2.5.4
surname	2.5.4.4
streetAddress	2.5.4.9
organizationName	attributeType.10

A set of Object Identifier tables in the ISODE `ETCDIR` directory are used to provide this mapping (see also Section 9 in *Volume One*). In general the default table will be sufficient, this section describes the tables for those who want to extend the default definitions.

This basic format is used to build up the specification of general object identifiers. This file is by default named *oidtable.gen*, and has simple mappings from string to oid. These strings can then be used in the definition of further oids (for example the "organizationName" oid in the table above).

This simple table is extended to give table formats for attributes This file is by default named *oidtable.at*. Each entry in this table is assumed to represent an attribute, so in addition to mapping strings onto oids, it also defines the syntax for that attribute:-

String representation	Object Identifier	Syntax
objectClass:	attributeType.0:	objectclass
aliasedObjectName:	attributeType.1:	dn
commonName,cn:	attributeType.3:	caseignorestring
searchGuide:	attributeType.14:	asn
x121Address:	attributeType.24:	numericstring
presentationAddress:	attributeType.29:	presentationaddress

This says that the attribute named "objectclass" represents the oid "attributeType.0" — using the expansion of "attributeType" (as defined in the file *oidtable.gen*) thus gives the oid "2.5.4.0". The syntax taken by the attribute value is "objectclass".

Similarly an "aliasedObjectName" has "dn" (DistinguishedName) syntax. The recognised syntaxes are defined in the BNF in Appendix B.

After the "syntax" an extra optional parameter is allowed; if this is the string "`file`" the the relevant attribute is designated a file attribute — see Section 14.7.4 for a description of file attributes.

The file *oidtable.oc* defines the object classes QUIPU knows about. The file again maps strings onto oids. Each string is assumed to represent an object class, and for each it defines the hierarchy (which object classes it is a subclass of), the mandatory attributes of the object class, and the optional attributes of the object class.

For example the entry

```
quipuObject:   quipuObjectClass.2 : top : acl : \
                masterDSA, slaveDSA, treeStructure
```

defines "quipuObject" to be an objectClass represented by the oid "quipuObjectClass.2". It is a SUBCLASS of the object class "top". An entry having this object class MUST have the attribute "acl", and MAY have the attributes "masterDSA", "slaveDSA" and "treeStructure" (NOTE The similarity between this definition and the ASN.1 definition of a "quipuObject" in Appendix C).

To allow for definitions of "attribute sets", there is a simple MACRO facility provided, for example, using the MACRO

```
localeAttributeSet = localityName, streetAddress ...
```

every occurrence of "`localeAttributeSet`" will be replace by the right hand side expression.

With the definition of the "string" form of the oid it is also possible to specify ONE abbreviation for the name. The standard tables use the following abbreviations:-

c	countryName
o	organizationName
ou	organizationalUnitName
cn	commonName
co	friendlyCountryName

The abbreviated name may be used anywhere the full name would be allowed.

Every entry stored in the QUIPU DSA is checked against these tables to see if the attributes supplied are allowed in the entry, to make sure the mandatory attributes are present, and the attributes themselves have the correct syntax.

If you wish to add you own definitions to the tables you should add them to the files *isode/dsap/oidtable.at.local*, *isode/dsap/oidtable.oc.local* and *isode/dsap/oidtable.gen.local* in that way, if a new set of tables are issued, you local entries will be preserved.

14.12 More Help Installing QUIPU

Contact “QUIPU-support”, the mail address is given in Section 2.6.

Chapter 15

Security Management

15.1 Configuration

There are a number of options that can be set when compiling QUIPU, these are discussed in Section 12.1. Three of the parameters have security implications:

NO_STATS: This option results in a DSA that doesn't record any usage and audit information. 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.

HAVE_PROTECTED: This option results in a DSA that can perform protected simple authentication.

HAVE_RSA: This option results in a DSA that can perform strong authentication.

To build a DSA that can perform strong or protected simple authentication, you will also need additional files that are not part of the ISODE. These files (the "QUIPU security upgrade") will soon be available and will be distributed by University College London. For further information, contact:

Steve Kille
Department of Computer Science
University College London

Gower Street,
London,
England

Internet: quipu-security@cs.ucl.ac.uk

15.1.1 QUIPU Userid

The DSA should not run under the userid of the super-user (*root*). The reason for this is that no application should be run as root unless it is absolutely necessary, just in case an undetected bug causes it to malfunction. (A process running as an ordinary user will be halted if it starts to misbehave; One running as root might carry on and delete important files, for example).

Ideally, the DSA should be assigned a special userid to run under. (Called, for example, "quipu"). If this is not possible, it should run under the userid of the DSA manager. Whichever option is taken, we shall henceforth refer to the selected userid as "quipu" in the documentation.

15.1.2 File Permissions

The EDB files are where QUIPU stores the contents of the Directory. As these files potentially contain sensitive information, they should be readable and writable only by the QUIPU userid.

There are two log files; one for debugging information and one for audit. Both of these files should be readable and writable only by the QUIPU userid. There is a potential problem here, in that ISODE's logging code usually makes log files writable by everybody. To be on the safe side, make a separate directory for QUIPU's logs (*/etc/isode/quipu-logs*, for example). Make this directory only accessible by the QUIPU userid, and change the tailoring parameter *logdir* so that the logs are written there.

15.2 Discretionary Access Control

The principles of discretionary access control are explained in the Chapter 11 of this manual. This section gives guidelines on setting the access control lists for entries in the DIT.

15.2.1 What must be Publicly Readable

Some attributes are used by the Directory itself for routing and other purposes. If these attributes are not publicly readable (and hence readable by all DSAs) then the Directory's internal communications may fail. If a DUA gives messages such as "Unavailable" this is one possible cause. Note that there are many other possible causes of such failures — Network congestion or a machine being down is the most likely explanation.

The following attributes ought to be publicly readable:

- masterDSA
- slaveDSA
- presentationAddress
- userCertificate
- treeStructure

The *userPassword* attribute ought to be comparable by everybody, but not readable (unless the entry is a spot shadowed DSA entry, in which case the userPassword must be publically readable, see Section ??).

15.3 Audit

15.3.1 Enabling Auditing

The *stats* parameter in the *quiputailor* file controls how much audit information is kept. It is advisable to enable recording of audit events at the *notice* level. More detailed information is given at the *trace* level. Both are enabled by default.

15.3.2 Relating Events to Users

Most events in the usage log contain an *association descriptor* instead of the name of the user who caused the event. An association descriptor is a (small) number which identifies a connection to QUIPU. (It is rather like knowing which terminal line a command came in on). To discover the user name, it is necessary to scan back through the log to find the record for the start of the association. This will contain the name of the user and how they authenticated themselves.

15.3.3 Format of Audit Records

Each record in the log file is formatted as follows:-

```
<AuditRecord> ::= <month> "/" <day> <time> <process>
                  <pid> "(" <userid> ")" <Event>
```

time: The time of the event in hh:mm:ss format.

process: The name of the program (quipu).

pid: The process id of the DSA.

userid: The id of the user who started the DSA running. It is *not* the id of the DUA which caused the event!

Event: is the rest of the message. The following sections describe most of the common messages.

15.3.4 Start of an Association

```
<BindEvent> ::= "Bind" "(" <Integer> ")"
                "(" <AuthType> ")"
                ":" <DN>
<AuthType> ::= "none" | "simple" | "protected" |
               "strong" | "noauth"
```

For example:

```
Bind (4) (simple): c=GB@o=University College
         London@ou=Computer Science@cn=Steve Kille
```

This means that Steve Kille has started using association descriptor 4, and proved his identity using simple authentication (i.e. a password).

15.3.5 End of an Association

```
<UnbindEvent> ::= "Unbind" "(" <Integer> ")" <WhoBy>
      ":" <DN> <WhoBy> ::= "(by this)" | "(by that)"
```

For example:

```
Unbind (4) (by that): c=GB@o=University College
           London@ou=Computer Science@cn=Steve Kille
```

This means that Steve Kille's DUA has disconnected from the DSA, and descriptor 4 is left free for use by someone else. The "(by that)" means that the DUA, rather than this DSA, decided to close the connection.

15.3.6 DAP Operation

```
<DAPOperationEvent> ::= <OpType> "(" <Integer> ")"
                        ":" <DN>
<OpType> ::= "Read" | "Search" | "List" | "Compare" |
             "Add" | "Remove" | "Modify" | "ModifyRDN" |
             "Getedb"
```

For example:

```
Read (4): c=gb@o=Nottingham University
```

This means that whoever is using association 4 (Steve Kille in this example) has read the entry for Nottingham University.

15.3.7 DAP Result

```
<DAPResultEvent> ::= "Result sent" "(" <Integer> ")"
<DAPErrorEvent>  ::= "Error sent"  "(" <Integer> ")"
```

Each operation will normally be answered by either a result or an error.

15.3.8 Chaining

```
<ChainingEvent> ::= "Chain" "(" <Integer> ")"
```

This means that the DSA has decided to contact another DSA in order to perform an operation received previously.

15.3.9 Updates

```
<UpdateEvent> ::= "Slave update" <DN> |
                  "Shadow update" <DN>
```

These indicate an EDB file or spot shadow entry have been updated.

15.3.10 Other Events

The are a few other less common messages that can be written to the audit log. The text messages should be self-explanatory.

15.3.11 Processing the Log Files

A script called *dsastats* may be used to process the log file and produce a report summarising the usage of a DSA. The report shows both the following:

- Who has accessed the DSA
- Which parts of the DIT have been accessed

The report summarises the calls received by the DSA according to the degree of authentication. This analysis is given for remote and local access. The report also tries to distinguish between system usage, which includes QUIPU getedb operations, DSA probing and testing by directory system and interface developers, and real usage. The ability to make this distinction rests on user names being supplied in bind requests. A large percentage of binds are currently anonymous — it is hoped that this report encourages directory administrators to install systems such that the provision of user names becomes the norm.

Configuring dsastats

The usefulness of the report relies on accurate configuration. The key configuration details to be noted are:

Local Organisation: Analysis of which parts of the DIT have been accessed is given by organisational unit for the pre-configured local organisation. For all other parts of the DIT, analysis is restricted to being by organisation.

Local addresses: A file should be set up (a skeleton is provided) which contains the leading substrings from local DUA and DSA addresses, as they appear in the quipu.log file. This file, which is called *quipulocaladds,* might look something like the following:

```
LOCAL-ETHER
Janet=000012345678
Internet=128.9
# file must end with this line
```

Filtering out system usage: A file should be set up (again a skeleton is provided) which contains the distinguished names of system users. This will probably include the names of directory software developers, directory administrators and dsa probes. Distinguished names can be specified case-independently. The file, which is called *quiputechusers,* might look something like the following:

```
c=GB@o=University College London@ou=Computer Science@cn=incads
c=GB@o=University College London@cn=DSA Probe
# File must end with this line
```

Running the script

The script can be configured to find the tailor files and log files. In this case, the script can be run by something like the following:

```
dsastats >usage.report
```

```
Summary of calls to DSA <giant armadillo>
From  2:48:46 on 04 feb to 13:24:09 on 07 feb

No. of binds                      local  remote

Anonymous                            63      82
Unauth name DAP                      17       3
Unauth name DSP                      22      14
Simple                                1       9

System usage (calls received)

Binds by Directory technicians                0
Reads of DSA entries                          0
Getedb operations                           953
Spot shadows                                  6

Who has used the directory?
*Real* usage by organisation
No. users No. binds
        1       145    anonymous
        1         5    c=gb, o=x-tel services ltd
        ...

Which parts of the Directory have been accessed - real usage?
No. ops  Subtree
Local subtree
     67  c=gb@o=university college london
     14  c=gb@o=university college london@ou=computer science
      1  c=gb@o=university college london@ou=genetics and biometry
      1  c=gb@o=university college london@ou=mathematics
      ...

Other parts of the DIT
      0  root
     66  c=gb
      1  c=gb@o=brunel university
      1  c=gb@o=edinburgh university
      1  c=gb@o=glasgow university
      ...

Which parts of the Directory have been accessed - system usage?
No. ops  Subtree
      ...
```

Figure 15.1: Example Output of dsastats

The above will not be possible if you run more than 1 DSA. The recommendation is then that each DSA logs to a file with a name of the form dsaname.usage. Usage would now be:

```
dsastats llama.usage >llama.report
```

A report produced by this script should resemble the example given in Figure 15.1.

Chapter 16

User Naming Architecture

16.1 Overview

The original work in INCA defined a naming architecture. Some of this work has been overtaken by standardisation activity, and other components are not relevant here. A few pieces are retained, as they may be of general use. The entire naming architecture is shown in Appendix C on page 293.

16.2 THORN

The THORN project is a significant ESPRIT project, which is implementing Directory services[FSiro88]. THORN has defined a Naming Architecture, which includes a number of non-standard, but useful attributes [SKill89c]. This architecture has been adopted for use in QUIPU at UCL, and for this reason, information in this architecture is not duplicated here. The latest version of the THORN Naming Architecture (also known as the RARE Naming Architecture) in available via FTAM from UCL (the address is given in the **Preface** of this manual) in the files *thorn/thorn-na.txt*, *thorn/thorn-na.ps* or *thorn/thorn-na.ms* depending whether you want a "plain text", "postscript" or "nroff -ms" version. Alternatively you could Mail `QUIPU-support` asking for the relevant document (see Section 2.6).

```
friendlyCountryName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
  caseIgnoreStringSyntax
          ::= {attributeType 8}
                      -- example "UK", "United Kingdom" etc.

friendlyCountry OBJECT-CLASS
  SUBCLASS OF country, quipuObject
  MUST CONTAIN { friendlyCountryName }
          ::= {objectClass 3}
```

Figure 16.1: Country

16.3 Common Name Forms

The use of ISO 3166 codes is abhorrent, as it makes the DUA perform things which the DSA should do. Therefore a more general country attribute is introduced (see Figure 16.1). The intention is to use this attribute to permit users to search for countries based on familiar strings.

QUIPU supports multiple inheritance, so there is a need to define an QUIPU Object class, which will permit all of the standard QUIPU attributes to be associated with all objects in the QUIPU Directory, the definition can be found in Figure C.1, in Appendix C.

16.4 DSA Naming Architecture

The following Naming Architecture components are needed in order to support the QUIPU Mechanism for distributed operations. These are defined here, as a convenient location. Numbers are assigned elsewhere in this manual.

```
QuipuObject ::= OBJECT−CLASS
        SUBCLASS OF top
        MUST CONTAIN {aCL}
        MAY CONTAIN {lastModifiedBy, lastModifiedTime, entrySecurityPolicy}

QuipuNonLeafObject ::= OBJECTCLASS
        SUBCLASS OF quipuObject
        MUST CONTAIN {masterDSA}
        MAY CONTAIN {slaveDSA, treeStructure, inheritedAttribute}

ExternalNonLeafObject ::= OBJECTCLASS
        SUBCLASS OF quipuObject
        MAY CONTAIN {subordinateReference, crossReference,
              nonSpecificSubordinateReference}

QuipuDSA ::= OBJECT−CLASS
    SUBCLASS OF dsa
    MUST CONTAIN { aCL, edbInfo, userPassword, manager, quipuVersion}
    MAY CONTAIN { description, lastModifiedBy, lastModifiedTime,
        dsaDefaultSecurityPolicy, dsaPermittedSecurityPolicy, relayDSA,
        listenAddress, info }


EDBInfoSyntax ::= SEQUENCE {
        edb           DistinguishedName,
        getFromDSA    DistinguishedName OPTIONAL,
                                  −− If omitted DSA is master
                                  −− Determine mode of update from this DSA
        sendToDSAs    NameList,
                                  −− Send these DSAs incremental updates
                                  −− Namelist is defined with the ACLs
        getEDBAllowed NameList
                                  −− List of DSAs allowed to pull EDB
        }

EdbInfo ::= ATTRIBUTE
    WITH ATTRIBUTE−SYNTAX EDBInfo
```

```
    MULTI VALUE

MasterDSA ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
                                -- Master QSR
                                -- Usually, but not necessarily single valued

SlaveDSA ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
                                -- Slave QSR

SubordinateReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint
    SINGLE VALUE

CrossReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint
    SINGLE VALUE

NonSpecificSubordinateReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint

QuipuVersion ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax

DSAControl ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
```

Figure 16.2: Naming Architecture

Part IV

Programmer's Guide

Chapter 17

Programming the Directory

This part of the manual is written for implementors who wish to access the Directory using the *libdsap*(3n) library. This might be an OSI application wishing to look up information, or an interactive Directory user-interface. The interface to the Directory is defined as a series of procedures which closely parallel the service defined in [CCITT88]. This section defines the procedures, and shows how they relate to the standard. To avoid unnecessary wordiness, the user is expected to be familiar with the standard.

17.1 Conventions

The library defines various C structures, described in the following sections. Associated with the C structures, are routines which:

- allocate structures
- free structures, including sub-structures
- compare structures
- copy structures
- print structures
- parse structures

The following conventions are followed in many routines, which are then briefly identified later. For a general structure x, there will be the following routines:

- Allocation:

  ```
  struct x *x_alloc ()
  ```

  Allocate memory for structure `x`.

- Freeing:

  ```
  x_free (a)
  struct x * a;
  ```

  Free structure `x`, and any nested structures.

  If the structure, `x`, is a linked-list type of structure, then as well as the routines described above, there will be routines of the form `x_comp_function` (where `function` is `alloc`, `free`, and so on). These act upon a single element of the list rather than the list as a whole.

- Copying:

  ```
  struct x * x_cpy (a)
  struct x *a;
  ```

  Return a copy of structure `a`.

- Comparing:

  ```
  int     x_cmp (a, b)
  struct x *a, *b;
  ```

  Return `0` if the structures `a` and `b` same. Return `-1` if `a<b`, or `1` if `a>b`. For some structures the notion of `a>b` and `a<b` is not defined, but will be consistent.

- Printing:

  ```
  int     x_print (ps, a, format)
  PS       ps;
  struct x *a;
  int      format;
  ```

Print the structure `a` to the presentation stream `ps`, using the specified format, either `READOUT`, for a pretty-printed format (for example the output of a showentry from *dish*); or, `EDBOUT`, for a format suitable for inclusion in an EDB file. The `PS` structure is described in *Volume One* of this manual.

For the structures representing a DN and RDN there is an extra format: `DIROUT`, which is used to get a representation of the name suitable for inclusion in a UNIX filename (essentially `EDBOUT` with the `@` symbols replaced with `/` separators).

- String format:

```
struct x * str2x (str)
char * str;
```

Most attribute syntaxes have a string representation. A string of the required syntax can be converted to the relevant c structure with the "str2x" function. Each of the string parsers assumes the input string has had multiple spaces and tabs removed, and replaced with single spaces. The function

```
char * Tidy_String (a)
char * a;
```

will do the relevant tidying if required. Strings printed with the `EDBOUT` format of x_print should be parsable.

NOTE `str2AttrV()` does not follow the "convention" and has an extra parameter, which is the expected syntax of the string.

The structures following the "convention" are listed below. The "prefix" column shows the value `x` should be replaced by to use the routines for the associated structure.

Structure	Prefix
RDN	rdn
DN	dn
AttributeType	AttrT
AttributeValue	AttrV
AV_Sequence	avs
Attr_Sequence	as
EntryInfo	entryinfo

17.2 Attributes

Attributes are fundamental components of the directory. Attributes have two parts, types and values, which are (respectfully) represented by two structures, `AttributeType` and `AttributeValue`.

To represent AttributeTypes, *libdsap*(3n) has a table (described in Section 14.11) which maps a string representation of attribute types onto OIDs.

This table must be loaded by calling

```
load_oid_table(name)
char * name;
```

The `name` parameter should be the name of the oidtable to load. For most purposes this should be "`oidtable`".

Before you load the table, any syntaxes you want special attribute handlers for should be loaded. To do this you will probably want to call

```
quipu_syntaxes()
```

which will load all the syntax handlers defined in Chapter 10. If not all there syntaxes are required, then a subset can be loaded by calling only the handlers required.

The call

```
dsap_init(argc,argv)
int *argc;
char *** argv;
```

can be used to load the oidtable for you, with the additional benifit that it will read the file */etc/dsaptailor* to find the location of the oidtable. The parameters it expects are the pointers to the `argc` and `argv` parameters passed to your "main()" routine. If a "`-c`" flag is found then this is used as the name of the DSA to contact as per dish (see Section 4.1.13). A "`-t`" can be used to specify which dsaptailor file to use, and "`-T`" to define an oidtable. As with loading the tables, the syntaxes you require should have been loaded before you make a call to `dsa_init` (Beware: `dsap_init` removes flags it interprets from the parameter list, if you care, pass a copy of them, or pass nulls!).

The oidtables are represented using the following structures:-

```
typedef struct {
        char    * ot_name;
        char    * ot_stroid;
        OID       ot_oid;
        OID       ot_aliasoid;
} oid_table;
#define NULLTABLE ((oid_table * )0)

typedef struct {
        oid_table oa_ot;
        short     oa_syntax;
} oid_table_attr, * AttributeType;
#define NULLTABLE_ATTR ((oid_table_attr *)0)
#define NULLAttrT ((oid_table_attr *)0)
```

The table consists of two fields, an integer syntax :- `oa_syntax`. This is an integer handle onto the abstract syntax of the attribute value. The value "`0`" indicates an unknown syntax hence "ASN.1" is used.

The second field `oa_ot` is a reference to a `oid_table` structure, which contain the oid — `ot_oid`, a numeric string representation of the oid — `ot_stroid`, and the local key string — `ot_name`.

The field `ot_aliasoid` is used to store an alternative OID for the same attribute. This is used to allow an attribute to change its OID over a period of time.

To create an `AttributeType` given a string representation of an OID, you should use

```
AttributeType str2AttrT (str)
char   *str;
```

which will look up the string supplied in the relevant OID tables and returns an `AttributeType` pointer into the oidtables.

There are various low level routines to manipulate entries in the OID table, however the `AttributeType` routines described in Section 17.1 will be sufficient for most uses, so the details of these "invisible" routines are omitted here.

Attribute values are represented by the structure `AttributeValue`:-

```
typedef struct attrVal {
        short       av_syntax;
        caddr_t     av_struct;
} * AttributeValue;

#define NULLAttrV (AttributeValue) NULL
```

The `AttributeValue` structure can hold attribute values in many formats. Different syntaxes use different specialised *C* structures pointed to by `av_struct`, the field `av_syntax` defines the syntax represented by the `av_struct` field. Unknown syntaxes are represented as a presentation element indicated by the syntax being zero. Section 17.4 describes this more fully.

Attribute values can now be linked together to form a multi-value attribute value. For this the `AV_Sequence` structure is used.

```
typedef struct avseqcomp {
    attrVal             avseq_av;
    struct avseqcomp    *avseq_next;
} avseqcomp, *AV_Sequence;

#define NULLAV ((AV_Sequence) 0)
```

This is simply a linked list of AttributeValues.

To create an `AV_Sequence` the two routines

```
AV_Sequence avs_comp_new (av)
AttributeValue av;
```

```
AV_Sequence avs_merge (a,b)
AV_Sequence a,b;
```

can be used.

Finally an attribute can now be formed by linking the attribute type, and multiple attribute values using the `AttrSequence` structure:-

```
typedef struct attrcomp {
    AttributeType       attr_type;
    AV_Sequence         attr_value;
    struct attrcomp     *attr_link;
    struct acl_info     *attr_acl;
} attrcomp, *Attr_Sequence;

#define NULLATTR ((Attr_Sequence) 0)
```

This structure is used singularly to represent an attribute, and as a linked list (linked using `attr_link`) to represent a set of attributes.

To create an `Attr_Sequence` the two routines

```
Attr_Sequence as_comp_new (at,as,acl)
AttributeType at;
AV_Sequence as;
struct acl_info * acl;

Attr_Sequence as_merge (a,b)
Attr_Sequence a,b;
```

can be used. The `acl` parameter is only used by the DSA, for use in a DUA it should be set to `NULLACL_INFO`.

The routine

```
Attr_Sequence as_combine (as,str)
AV_Sequence as;
char * str;
```

is also provided, as an optimisation of

```
as_merge(as,str2as(str));
```

The approach to access control is defined in [SKill89b]. The representation of the attribute for access control is given by the structure shown below. This closely follows the ASN.1 definition.

```
struct acl_info {
    int              acl_categories;
#define ACL_NONE        0
#define ACL_DETECT      1
#define ACL_COMPARE     2
#define ACL_READ        3
#define ACL_ADD         4
#define ACL_WRITE       5
    int              acl_selector_type;
#define ACL_ENTRY       0
#define ACL_OTHER       1
#define ACL_PREFIX      2
#define ACL_GROUP       3
    DN               acl_name;
    struct acl_info *acl_next;
};
```

The DN `acl_name` is used for the type `ACL_PREFIX` and `ACL_OTHER` only.

17.3 Distinguished Names

The structures which define names are critical. Names are essentially made up of attributes, but because of their importance, a different — more specific structure is used.

```
typedef struct rdncomp {
    AttributeType       rdn_at;
    attrVal             rdn_av;
    struct rdncomp      *rdn_next;
} rdncomp, *RDN;

#define NULLRDN ((RDN) 0)
```

An RDN consists of a set of Attribute Types (`rdn_at`) and Attribute Values (`rdn_av`), the set is linked using `rdn_next`.

RDNs can the be joined in a sequence to form a Distinguished Name (DN):-

```
typedef struct dncomp {
    RDN                 dn_rdn;
    struct dncomp       *dn_parent;
} dncomp, *DN;

#define NULLDN ((DN) 0)
```

The following routines are provided in addition to the routines defined is Section 17.1 to allow the user to create and manipulate names

- Create a DN given an RDN:

  ```
  dn_comp_new (rdn)
  RDN     rdn;
  ```

- Append a DN to the end of another DN:

  ```
  dn_append (dn1, dn2)
  DN      dn1,
          dn2;
  ```

  `dn2` is added to `dn1`.

- Create a RDN given and AttributeType and AttributeValue:

  ```
  RDN rdn_comp_new (at, av)
  AttributeType  at;
  AttributeValue av;
  ```

- Merge two RDNs:

  ```
  RDN rdn_merge (rdn1, rdn2)
  RDN     rdn1,
          rdn2;
  ```

  RDN's form a set, to make comparisons easier, this implementation assumes them to be ordered sets. This routine takes two ordered RDN sets, and merges them. If RDNs received across the network are not ordered, the decoding process will order them for you.

17.3.1 User Friendly Naming

There are various routines to implement User Friendly Naming (UFN)[SKill90]. These are briefly described in this section.

To print a DN to a PStream as a UFN you can use

```
ufn_dn_print (ps,dn,multiline)
PS  ps;
DN  dn;
int multiline;
```

If the `multiline` option is `TRUE` the UFN will be printed on multiple lines, otherwise, a single line of output will be used. The `ufn_dn_print` routine prints each RDN is printed using

```
ufn_rdn_print (ps,rdn)
RDN  rdn;
PS   ps;
```

Parsing a UFN into a DN is a little more complex:

```
ufn_match (argc,argv,interact,result,el)
int argc;
char ** argv;
DNS (* interact) ();
DNS * result;
envlist el;
```

The `argc` and `argv` parameters provide the components of the UFN, which will typically have been generated from a comma separated UFN string. The `el` component is used to pass a UFN environment list to the lookup routine. If this is left as NULL, this is generated from the system wide *ufnrc* file.

A sequence of DNs is returned in the `result` paramenter. `DNS` has the following structure:

```
typedef struct dn_seq {
    DN  dns_dn;
    struct dn_seq *dns_next;
} *DNS;
```

The `interact` parameter if the name of a routine to interact with the user if the UFN process is unclear as to which name to choose.

```
DNS interact (dns,dn,s)
DNS dns;
DN dn;
char * s;
```

The `dns` parameter is the sequence of potential names, `dn` is the DN of the environment below which the names have been, `s` is UFN element being matched. The user should be asked to select which DNs should be used, and the DNs returned. DNs not required in the `dns` passed in should be freed. If user interaction is not required, as simple interact routine to return a NULL DNS should be provided.

17.4 Adding New Syntaxes to QUIPU

It will probably be necessary to add knowledge of extra syntaxes to Directory User Interfaces as the range of applications grow. QUIPU has been designed to make this as easy as possible.

This section describes how you might do this, by adding knowledge of a `FOOBAR` syntax to *libdsap*(3n) library. `FOOBAR` is represented by the c structure "`struct foobar {...};`".

The DSA does not need to know about the new structure, as it will handle unknown syntaxes as blocks of ASN.1.

The procedure call

```
add_attribute_syntax (sntx,enc,dec,parse,
                      print,cpy,cmp,sfree,
                      print_pe,approx,multiline)
char *  sntx;
IFP     enc,dec,parse,print,cpy,cmp,sfree,approx;
char *  print_pe;
char    multiline;
```

is used to add knowledge of new attribute syntaxes to QUIPU.

The parameters are used as follows:-

`sntx`: The string defining the syntax, in this case it should be `Foobar`. Any attributes defined as having the syntax "Foobar" in the oidtables will the be handled using the following routines.

`enc`: This is a function pointer to a routine that will convert a C structure representation of foobar into a presentation element that can be sent across the network. The routine is given a single parameter — a pointer to the foobar structure, and is expected to return a PE. You are reminded of the *pepy*(1c), *posy*(1c) and *pepsy*(1c) utilities described in *Volume Four* of this manual, they may be of help in creating this encoder.

`dec`: This performs the inverse of the above, it is passed a PE, and should return a C structure representation.

`parse`: Each syntax needs to have a representation that can be read from and written to an EDB file. This defined routine should take a single `char *` argument and return a C structure representation.

`print`: This routine is used to print the syntax to a `PStream`[1]. The arguments are:

```
foobar_print (ps,fb,format)
PS ps;
struct foobar * fb;
int format;
```

If `format` is `READOUT` the structure should be printed to the stream PS in a "human readable" manner, otherwise is should be printed in a form that can be parsed by the "parse" function. NOTE in many case both formats will be the same.

`cpy`: This function should take a foobar structure as a parameter and returns a copy of it.

`cmp`: A function that takes two foobar structures and compares them — returning 0 if they are the same; 1 if the first is considered "greater" than the second; –1 otherwise. If there is no appropriate ordering for the syntax, return 2. If an error occurs during the comparison return –2.

`sfree`: A routine to free the structure foobar.

[1]`PStream`'s are discussed in *Volume One* of this manual.

print_pe: The name of an external process to "exec" when the `READOUT` option is supplied to the print routine. This is generally set to `NULLCP` which means the default "print" routine is used.

approx: A routine to perform approximate matching (if required). The routine should expect two parameters as shown:

```
foobar_approx (a,b)
    struct filter_item * a;
    AV_Sequence b;
```

The routine should return `OK` or `NOTOK` depending on whether any attribute value in the `AV_Sequence` approximately matches the `filter_item`.

multiline: If `TRUE` the multi-value attributes will be printed on separate lines:

```
foobar = value1
foobar = value2
```

otherwise the attribute will be printed on one line:

```
foobar = value1 & value2
```

17.4.1 Where to Add the Syntax Definition

Where should you add this procedure call, to define the new syntaxes?

This depends on which applications you want to know about the syntax. If you want the DSA and all DUAs to know about the syntax, then this call should be added to libdsap, in the directory dsap/common. The file dsap/common/quipu_sntx.c is where other similar calls are made. A small test program `test` is included in the common directory, and is made with "`make test`". It takes as input a string representation of attributes, and exercises the handlers. If this works for your new attribute, then it should be safe to re-compile the DSA/DUA and try them.

If you only wish your DUAs to know about the new syntax, then add the call into you DUAs code, before loading the oidtables or calling `dsap_init`.

Chapter 18

The Procedural DUA

The *libdsap*(3n) library defines a set of procedure calls which correspond to each of the X.500 abstract operations. This chapter describes those procedure calls.

18.1 Procedure Model

Each operation is accessed via a procedure call of the same name as the X.500 operation, prefixed by "`ds_`". The procedure is supplied an argument structure, and returns either an error or result structure. For example the `read` operation is invoked by calling `ds_read (argument,error,result)`.

The return value of the procedures have the following common values, which indicated whether an error or result is returned:-

`DS_OK`: Operation completed successfully, the result structure will have the corresponding result (if the operation generates results).

`DS_ERROR_LOCAL`: Error within the DUA module.

`DS_ERROR_CONNECT`: Failed to connect to a remote DSA.

`DS_ERROR_PROVIDER`: Other OSI provider error.

`DS_ERROR_REMOTE`: Remote error. Further details will be in the error parameter in the procedure call.

`DS_X500_ERROR`: This is the same as `DS_ERROR_REMOTE`.

These values defined in *quipu/ds_error.h*, which must be included in your program (there are other return values defined here, but these should only occur within the DSA, and should not be returned by a DUA call).

18.2 Common Parameters

All of the DAP operations described in Sections 18.5 to 18.11.4 have certain common parameters in their arguments and results. These structures are now described.

18.2.1 Arguments

The common arguments are represented by the following structure:-

```
typedef struct common_args {
    ServiceControl        ca_servicecontrol;
    DN                    ca_requestor;
    struct op_progress    ca_progress;
    int                   ca_aliased_rdns;
#define CA_NO_ALIASDEREFERENCED -1
    struct security_parms * ca_security;
    struct signature    * ca_sig;
    struct extension    * ca_ext;
} CommonArgs;
```

The fields `ca_ext`, `ca_progress`, `ca_requestor` and `ca_aliased_rdns` are provided as they are defined within X.500. Neither the QUIPU DSA or DUA use these fields.

The field `ca_servicecontrol` is used to select the type of service the DSA should provide, the structure is given below

```
typedef struct svccontrol {
    int         svc_options;
#define SVC_OPT_PREFERCHAIN             0X001
#define SVC_OPT_CHAININGPROHIBIT        0X002
#define SVC_OPT_LOCALSCOPE              0X004
#define SVC_OPT_DONTUSECOPY             0X008
#define SVC_OPT_DONTDERFERERENCEALIAS   0X010
```

```
    char        svc_prio;
#define SVC_PRIO_LOW    0
#define SVC_PRIO_MED    1
#define SVC_PRIO_HIGH   2
    int         svc_timelimit;
#define SVC_NOTIMELIMIT -1
    int         svc_sizelimit;
#define SVC_NOSIZELIMIT -1
    int         svc_scopeofreferral;
#define SVC_REFSCOPE_NONE       -1
#define SVC_REFSCOPE_DMD        0
#define SVC_REFSCOPE_COUNTRY    1
} svccontrol, ServiceControl;
```

The values takes by each field within the ServiceControl structure are described in full in [CCITT88].

18.2.2 Results

```
typedef struct common_results {
    DN      cr_requestor;
    char    cr_aliasdereferenced;
    struct security_parms  * cr_security;
    struct alg_id          * cr_alg;
    char                   * cr_tmp;
    int                      cr_len;
} common_results, CommonResults;
```

The filed `cr_aliasdereferenced` is set to `TRUE` if the base object of the operation was an alias, and was dereferenced.

The other fields are not used by the current version of QUIPU.

18.3 Continuation References

"Continuation References" are returned by the directory when the operation asked for could not be fully completed, in which case the structure `ContinuationRef` is returned as part of the error or results structures. The structure is explained below:-

```
typedef struct continuation_ref {
    DN          cr_name;
    struct op_progress cr_progress;
    int         cr_rdn_resolved;
#define CR_RDNRESOLVED_NOTDEFINED  -1
    int         cr_aliasedRDNs;
#define CR_NOALIASEDRDNS -1
    int         cr_reftype;
#define RT_UNDEFINED    -1
#define RT_SUPERIOR     1
#define RT_SUBORDINATE  2
#define RT_CROSS        3
#define RT_NONSPECIFICSUBORDINATE 4
    struct access_point * cr_accesspoints;
    struct continuation_ref *cr_next;
}continuation_ref, *ContinuationRef;
```

`cr_name`: The DN that has only been partially explored.

`cr_rdn_resolved`: The number of RDNs in the name that have been examined — hence how far down the DIT the query has been taken.

`cr_aliasedRDNs`: If `TRUE` then some of the RDNs were aliases.

`cr_reftype`: The type of reference that was used to get this information.

`cr_accesspoints`: The access point of the DSA to contact. The structure for an access point is as follows:-

```
struct access_point {
    DN                  ap_name;
    struct PSAPaddr     *ap_address;
    struct access_point *ap_next;
};
```

`ap_name`: The DN of the DSA to contact.

`ap_address`: The address of the DSA (derivable from the name).

`ap_next`: There may be more than one access point, hence they form a linked list structure.

cr_next: There may be more than one access point, if so they form a linked list structure.

18.4 Errors

All DAP operations return a common error structure, which maps closely onto the service definitions. An error is only returned if the operation failed, if successful (indicated by the return value DS_OK) the error structure returned is undefined.

```
struct DSError {
    int dse_type;
#define DSE_INTR_ABANDON_FAILED -5
#define DSE_INTR_ABANDONED      -4
#define DSE_INTRERROR           -3
#define DSE_LOCALERROR          -2
#define DSE_REMOTEERROR         -1
#define DSE_NOERROR             0
#define DSE_ATTRIBUTEERROR      1
#define DSE_NAMEERROR           2
#define DSE_SERVICEERROR        3
#define DSE_REFERRAL            4
#define DSE_ABANDONED           5
#define DSE_SECURITYERROR       6
#define DSE_ABANDON_FAILED      7
#define DSE_UPDATEERROR         8
#define DSE_DSAREFERRAL         9
    union {
       struct DSE_attribute    dse_un_attribute;
       struct DSE_name         dse_un_name;
       struct DSE_service      dse_un_service;
       struct DSE_referral     dse_un_referral;
       struct DSE_abandon_fail dse_un_abandon_fail;
       struct DSE_security     dse_un_security;
       struct DSE_update       dse_un_update;
    }   dse_un;
};
```

The field `dse_type` is used to indicate what sort of error has occurred, and hence which structure from the union is used.

The value `DSE_LOCALERROR` is used to indicate that the error came from within the DUA, there is no associated structure in the union for this error.

The value `DSE_REMOTEERROR` is used to indicate that an error occurred at the DSA end, and the request was rejected, again there is no associated structure in the union for this error.

The structures in the union for the other error conditions are as described in the following sections.

18.4.1 Attribute Error

```
struct DSE_attribute {
    DN DSE_at_name;
    struct DSE_at_problem DSE_at_plist;
};
```

`DSE_at_name`: The name of the entry causing the error.

`DSE_at_plist`: A list of the errors:-

```
struct DSE_at_problem {
    int                     DSE_at_what;
#define DSE_AT_NOSUCHATTRIBUTE          1
#define DSE_AT_INVALIDATTRIBUTESYNTAX   2
#define DSE_AT_UNDEFINEDATTRIBUTETYPE   3
#define DSE_AT_INAPPROPRIATEMATCHING    4
#define DSE_AT_CONSTRAINTVIOLATION      5
#define DSE_AT_TYPEORVALUEEXISTS        6
    AttributeType           DSE_at_type;
    AttributeValue          DSE_at_value;
    struct DSE_at_problem * dse_at_next;
};
#define DSE_AT_NOPROBLEM ((struct DSE_at_problem*)0)
```

The fields are used as follows:-

`DSE_at_what`: Indicates which error has occurred.

`DSE_at_type`: The attribute type causing the error.

DSE_at_value: The associated value (if any).

dse_at_next: There may be more than one such error — if so they form a linked list.

18.4.2 Name Error

```
struct DSE_name {
    int DSE_na_problem;
#define DSE_NA_NOSUCHOBJECT            1
#define DSE_NA_ALIASPROBLEM            2
#define DSE_NA_INVALIDATTRIBIBUTESYNTAX 3
#define DSE_NA_ALIASDEREFERENCE        4
    DN  DSE_na_matched;
};
```

`DSE_na_matched` is the part of the DN successfully matched.

18.4.3 Referral Errors

```
struct DSE_referral {
    ContininuationRef DSE_ref_candidates;
    DN                DSE_ref_prefix;
};
```

The continuation reference supplies information on how to continue the query. The structure is described in Section 18.3

The field `DSE_ref_prefix` is for DSP only.

You should generally chase such referrals.

18.4.4 Security Error

```
struct DSE_security {
    int DSE_sc_problem;
#define DSE_SC_AUTHENTICATION          1
#define DSE_SC_INVALIDCREDENTIALS      2
#define DSE_SC_ACCESSRIGHTS            3
#define DSE_SC_INVALIDSIGNATURE        4
#define DSE_SC_PROTECTIONREQUIRED      5
```

```
#define DSE_SC_NOINFORMATION            6
};
```

18.4.5 Service Error

```
struct DSE_service {
    int DSE_sv_problem;
#define DSE_SV_BUSY                             1
#define DSE_SV_UNAVAILABLE                      2
#define DSE_SV_UNWILLINGTOPERFORM               3
#define DSE_SV_CHAININGREQUIRED                 4
#define DSE_SV_UNABLETOPROCEED                  5
#define DSE_SV_INVALIDREFERENCE                 6
#define DSE_SV_TIMELIMITEXCEEDED                7
#define DSE_SV_ADMINLIMITEXCEEDED               8
#define DSE_SV_LOOPDETECT                       9
#define DSE_SV_UNAVAILABLECRITICALEXTENSION 10
#define DSE_SV_OUTOFSCOPE                       11
#define DSE_SV_DITERROR                         12
};
```

18.4.6 Update Error

```
struct DSE_update {
    int DSE_up_problem;
#define DSE_UP_NAMINGVIOLATION          1
#define DSE_UP_OBJECTCLASSVIOLATION     2
#define DSE_UP_NOTONNONLEAF             3
#define DSE_UP_NOTONRDN                 4
#define DSE_UP_ALREADYEXISTS            5
#define DSE_UP_AFFECTSMULTIPLEDSAS      6
#define DSE_UP_NOOBJECTCLASSMODS        7
};
```

18.4.7 Abandon Failure

If an abandon operation fails then the following is used:-

```
struct DSE_abandon_fail {
```

```
    int DSE_ab_problem;
#define DSE_AB_NOSUCHOPERATION  1
#define DSE_AB_TOOLATE          2
#define DSE_AB_CANNOTABANDON    3
    int DSE_ab_invokeid;
};
```

18.4.8 Error Handling Procedures

The *libdsap*(3n) library has three routines for handling errors.

```
ds_error (ps,error)
PS ps;
struct DSError * error;
```

This routine will take the error, and pretty-print the contents to the PStream `ps`.

```
log_ds_error (error)
struct DSError * error;
```

This routine will take the error, print a simple message in "dsap.log" at "LLOG_EXCEPTIONS" logging level, and a more detailed report at "LLOG_TRACE" logging level.

```
ds_error_free (err)
struct DSError * err;
```

This frees the embedded structures within the error structure, BUT NOT `DSError` itself.

18.5 Binding and Unbinding

Before operations may be invoked, the DUA must BIND to the DSA.

The `bind` procedure

```
secure_ds_bind (arg, error, result)
    struct ds_bind_arg        * arg;
    struct ds_bind_arg        * result;
    struct ds_bind_error      * error;
```

is used for this purpose.

You will need to include *quipu/bind.h* to use this procedure.

The `ds_bind_arg` structure is defined as follows.

```
struct ds_bind_arg {
    int  dba_version;
#define DBA_VERSION_V1988 0
    int dba_auth_type;
#define DBA_AUTH_NONE      0
#define DBA_AUTH_SIMPLE    1
#define DBA_AUTH_STRONG    2
#define DBA_AUTH_EXTERNAL  3
#define DBA_AUTH_PROTECTED 4
    char  *dba_time1;
    char  *dba_time2;
    struct random_number dba_r1;
    struct random_number dba_r2;
    DN     dba_dn;
    int    dba_passwd_len;
#define DBA_MAX_PASSWD_LEN 512
    char   dba_passwd[DBA_MAX_PASSWD_LEN];
    struct signature        *dba_sig;
    struct certificate_list *dba_cpath;
    char * dba_vtmp;
    int    dba_vlen;
};
```

dba_version: should always be set to `DBA_VERSION_V1988`.

dsa_auth_type: is used to select the type of authentication required.

dba_dn: is the name of the entity you wish to bind as, this can be `NULLDN` if you only require access to "publically readable" information.

dba_passwd: The password required to bind as the entity. This will be checked against the "userPassword" attribute in the entity.

dba_passwd_len: The length of the string in `dsa_passwd`.

dba_sig: The strong authentication signature (which must be provided if strong authentication is used in the bind call).

dba_cpath: The strong authentication certification path (optional).

dba_time1: Used for protected and strong authentication.

dba_time2: Optionally used for protected simple authentication.

dba_r1: Random number used for protected and strong authentication.

dba_r2: Optionally used for protected simple authentication.

dba_vtmp: Unused

dba_vlen: Unused

The `bind` operation will try to connect to the DSA, whose address is defined in the external `char *`, "`dsa_address`", the syntax of the string is that of an ISODE PSAP address.

If `secure_ds_bind()` returns `DS_OK`, then you have a successful connection, otherwise a `ds_bind_error` structure will inform you of the error condition.

```
struct ds_bind_error {
    int  dbe_version;
    char dbe_type;
#define DBE_TYPE_SERVICE  1
#define DBE_TYPE_SECURITY 2
    int  dbe_value;
};
```

The filed `dbe_value` takes a value as defined for a `DSE_security` or `DSE_service` error shown in Sections 18.4.4 and 18.4.5.

18.5.1 No Authentication

To bind without using any authentication, `dsa_auth_type` should be set to `DBA_AUTH_NONE`. The fields `dba_version` and `dba_dn` must be filled in.

18.5.2 Simple Authentication

To bind using simple authentication, the `dsa_auth_type` field should be set to the value `DBA_AUTH_SIMPLE`. The `dba_version`, `dba_dn`, `dba_passwd` and `dba_passwd_len` fields must be filled in.

For backwards compatibility the routine

```
ds_bind (arg, error, result)
    struct ds_bind_arg          * arg;
    struct ds_bind_arg          * result;
    struct ds_bind_error        * error;
```

is still supplied, and gives you an unprotected simple bind request.

18.5.3 Protected Simple Authentication

To bind using protected simple authentication, the `dsa_auth_type` field should be set to to the value `DBA_AUTH_PROTECTED`. The fields `dba_version`, `dba_dn`, `dba_time1`, `dba_r1`, `dba_passwd` and `dba_passwd_len` must be filled in. The fields `dba_time2` and `dba_r2` must be either filled in or set to `NULL`.

18.5.4 Strong Authentication

To bind using strong authentication, the `dsa_auth_type` field should be set to the value `DBA_AUTH_STRONG`. The fields `dba_version`, `dba_dn`, `dba_time1`, `dba_r1`, and `dba_sig` must be filled in. The field `dba_cpath` must be either filled in or set to `NULL`.

18.6 Unbind

The unbind operation has no parameters.

```
ds_unbind ()
```

and is used to disconnect from the directory.

18.7 Read

The read operation is used to access information from a particular entity in the DSA.

```
ds_read (arg, error, result)
    struct ds_read_arg          * arg;
    struct ds_read_result       * result;
    struct DSError              * error;
```

You will need to include *quipu/read.h* to use this procedure.
The argument `ds_read_arg` is used to formulate the DAP request:-

```
struct ds_read_arg {
    CommonArgs         rda_common;
    DN                 rda_object;
    EntryInfoSelection rda_eis;
};
```

The parameters are used as follows:-

`rda_common`: The common arguments as described in Section 18.2.1

`rda_object`: The DN of the object you want to read

`rda_eis`: The "Entry Information Selection", which defines which attributes you want to be returned, this is defined in Section 18.7.1

The results returned on a `DS_OK` return form the structure shown below:-

```
struct ds_read_result {
    CommonResults rdr_common;
    EntryInfo     rdr_entry;
};
```

`rdr_common`: The common results as described in Section 18.2.2

`rdr_entry`: This is a pointer to the "Entry Information" which contains the attributes you requested. This structure is defined in Section 18.7.2.

18.7.1 Entry Information Selection

Operations that return an Entry Information structure, will have an "Entry Information Selection" in their arguments to specify exactly what the DSA should return. The structure is represented thus:-

```
typedef struct {
    char                eis_allattributes;
    Attr_Sequence       eis_select;
    char                eis_infotypes;
#define EIS_ATTRIBUTETYPESONLY  0
#define EIS_ATTRIBUTESANDVALUES 1
} EntryInfoSelection;
```

The parameters are used as follows:-

eis_allattributes: If TRUE the all attributes are required.

eis_select: If eis_allattributes is FALSE then this field is used to specify the attributes you want. ONLY the attribute types of this structure need to be set, any attribute values will be ignored.

eis_infotypes: If EIS_ATTRIBUTETYPESONLY is set, then only attribute types will be returned, otherwise the values will also be returned.

18.7.2 Entry Information

The structure "Entry Information" is used by some of the operations to return data to the DUA.

```
typedef struct entrystruct {
    DN                  ent_dn;
    Attr_Sequence       ent_attr;
    int                 ent_iscopy;
#define INFO_MASTER 0x001
#define INFO_COPY   0x002
#define INFO_CACHE  0x003
    char                ent_pepsycopy;
    time_t              ent_age;
    struct entrystruct  *ent_next;
} entrystruct, EntryInfo;
```

The parameters are used as follows:-

ent_dn: The DN of the entry.

ent_attr: The requested attributes of the requested entry.

ent_iscopy: This is either INFO_MASTER or INFO_COPY indicating whether master data or copied data was used to generate the results. INFO_CACHE is not applicable to a DUA.

ent_pepsycopy: Unused.

ent_age: This field is the age of the entry, used to expire cached information.

ent_next: There may be many results, this is used to link them.

18.8 Compare

The compare operation is used to compare an asserted attribute, with an attribute in the directory.

```
ds_compare (arg, error, result)
    struct ds_compare_arg       * arg;
    struct ds_compare_result    * result;
    struct DSError              * error;
```

You will need to include *quipu/compare.h* to use this procedure.
The argument is as follows:-

```
struct ds_compare_arg {
    CommonArgs cma_common;
    DN         cma_object;
    AVA        cma_purported;
};
```

cma_common: The common arguments as described in Section 18.2.1.

cma_object: The DN of the object you want to compare an attribute against.

`cma_purported`: The attribute you want to compare. This structure is defined in Section 18.8.1.

If successful, the result structure is as follows:-

```
struct ds_compare_result {
    CommonResults cmr_common;
    DN            cmr_object;
    char          cmr_matched;
    char          cmr_iscopy;
    char          cmr_pepsycopy;
    time_t        cmr_age;
};
```

`cmr_common`: The common results as described in Section 18.2.2.

`cmr_object`: The DN of the entry the attribute was compared against. This may not be the same as the DN supplied in the argument (e.g., if an alias was dereferenced).

`cmr_matched`: If `TRUE` then the attributes were the same, otherwise they were different.

`cmr_iscopy`: If `TRUE` then the compare took place against a copy of the attribute.

`cmr_pepsycopy`: Unused.

`cmr_age`: Unused.

18.8.1 Attribute Value Assertion

An "Attribute Value Assertion" is used by some of the operations to specify an attribute type/value pair, the structure used in this case is shown below.

```
typedef struct {
    AttributeType  ava_type;
    AttributeValue ava_value;
}AVA;
```

18.9 List

The list operation is show below. This returns a list of subordinates of the specified entry.

```
ds_list (arg, error, result)
    struct ds_list_arg          * arg;
    struct ds_list_result       * result;
    struct DSError              * error;
```

You will need to include *quipu/list.h* to use this procedure.

```
struct ds_list_arg {
    CommonArgs lsa_common;
    DN         lsa_object;
};
```

`lsa_common`: The common arguments as described in Section 18.2.1.

`lsa_object`: The DN of the object you want to list the children of.

The results of a successful list form the following structure:-

```
struct ds_list_result {
    CommonResults        lsr_common;
    DN                   lsr_object;
    time_t               lsr_age;
    struct subordinate * lsr_subordinates;
    POQ                  lsr_poq;
    struct ds_list_result * lsr_next;
};
```

`lsr_common`: The common results as described in Section 18.2.2.

`lsr_object`: The DN of the entry whose children were listed. This may not be the same as the DN supplied in the argument (e.g., if an alias was dereferenced).

`lsr_age`: Not currently used.

lsr_subordinates: This structure contains the RDNs that were found below the base object. The structure is as follows:-

```
struct subordinate {
    RDN                    sub_rdn;
    char                   sub_aliasentry;
    char                   sub_copy;
    struct subordinate *  sub_next;
};
```

sub_rdn: The RDN of this child.

sub_aliasentry: If TRUE then it is an alias entry.

sub_copy: If TRUE then the list took place against a copy of the object.

sub_next: A pointer to the next element in the list.

lsr_poq: A partial outcome qualifier, the structure is as follows:

```
#define LSR_NOLIMITPROBLEM      -1
#define LSR_TIMELIMITEXCEEDED   0
#define LSR_SIZELIMITEXCEEDED   1
#define LSR_ADMINSIZEEXCEEDED   2
typedef struct {
   int             poq_limitproblem;
   ContinuationRef poq_cref;
   char            poq_no_ext;
} POQ;
```

poq_limitproblem: Used to indicates which type of limit has been reached.

poq_cref: A list of continuation references. Continuation References are described in Section 18.3.

poq_no_ext: Not used

lsr_next: A pointer to the next element if the list is uncorrelated. QUIPU will always return correlated results..

18.10 Search

The search operation performs a key Directory functionality. This is done on the basis of filters as described in Section 18.10.1 below The operation is defined as:-

```
ds_search (arg, error, result)
    struct ds_search_arg        * arg;
    struct ds_search_result     * result;
    struct DSError              * error;
```

You will need to include *quipu/ds_search.h* to use this procedure.

```
struct ds_search_arg {
    CommonArgs         sra_common;
    DN                 sra_baseobject;
    int                sra_subset;
#define SRA_BASEOBJECT          0
#define SRA_ONELEVEL            1
#define SRA_WHOLESUBTREE        2
    Filter             sra_filter;
    char               sra_searchalias;
    EntryInfoSelection sra_eis;
};
```

sra_common: The common arguments as described in Section 18.2.1.

sra_baseobject: The DN of the object you want to start the search from.

sra_subset: This specifies which part of the DIT to search.

sra_filter: The filter to apply to the searched entries to see if the entry is required. Filters are discussed in Section 18.10.1

sra_searchalias: If TRUE aliases encountered below the search baseobject are dereferenced, and the dereferenced object searched. Note how this is different to the service control "dontderferencealiases" which is used to control dereferencing of entities above the base object.

sra_eis: The “Entry Information Selection”, which defines which attributes you want to be returned (if the filter is matched), this is defined in Section 18.7.1.

The results form the following structure:-

```
struct ds_search_result {
    char  srr_correlated;
    union {
        struct ds_search_unit   * srr_unit;
        struct ds_search_result * srr_parts;
    }     srr_un;
    struct ds_search_result * srr_next;
};
```

srr_correlated: If `TRUE` the the results are said to be correlated, that is, there will only be one element in the list of search results. A DSA may return uncorrelated result, in which case the routine

```
correlate_search_results(sr_res)
struct ds_search_result * sr_res;
```

can be called, which will correlate the results.

NOTE QUIPU DSAs will always return correlated results.

srr_un: If the results are uncorrelated then the union will contain a nested `ds_search_result` structure, which contains a list of the uncorrelated results.

If the results are correlated, then this union will contain a pointer to the correlated results. These form the structure

```
struct ds_search_unit {
    CommonResults   srr_common;
    DN              srr_object;
    EntryInfo     * srr_entries;
    POQ             srr_poq;
};
#define CSR_common       srr_un.srr_unit->srr_common
#define CSR_object       srr_un.srr_unit->srr_object
```

```
#define CSR_entries      srr_un.srr_unit->srr_entries
#define CSR_limitproblem srr_un.srr_unit->\
                             srr_poq.poq_limitproblem
#define CSR_cr           srr_un.srr_unit->\
                             srr_poq.poq_cref
```

NOTE the `#define`s to access the elements of this structure. The fields are used as follows:

`srr_common`: The common results as described in 18.2.2.

`srr_object`: The DN of the entry used as the base object of the search. This may not be the same as the DN supplied in the argument (e.g., if an alias was dereferenced).

`srr_entries`: A pointer to the "Entry Information", which will contain the attributes you requested (if present in the entry). The `EntryInfo` structure is defined in Section 18.7.2.

`srr_poq`: A partial outcome qualifier. This is the same as for the list operation described in the previous section.

`srr_next`: This is a pointer to the next result in a list of uncorrelated results.

18.10.1 Filters

A filter in its simplest sense is a single `filter_item`. This is used to perform one of six basic tests on one attribute of an entry. If the "match" is good, then the entry is returned as part of the search result structure. The six basic types of `filter_item` are represented by the structure below:-

```
struct filter_item {
    int         fi_type;
#define FILTERITEM_EQUALITY       1
#define FILTERITEM_SUBSTRINGS     2
#define FILTERITEM_GREATEROREQUAL 3
#define FILTERITEM_LESSOREQUAL    4
#define FILTERITEM_PRESENT        5
#define FILTERITEM_APPROX         6
    union {
```

```
        AttributeType     fi_un_type;
        AVA               fi_un_ava;
        Filter_Substrings fi_un_substrings;
    }           fi_un;
    IFP         fi_ifp;
};
```

fi_type: Defines the type of `filter_item` being represented.

fi_un.fi_un_type: `FILTERITEM_PRESENT` matches, just supply an AttributeType, an entry matches if this attribute exists within the entry.

fi_un.fi_un_ava: `FILTERITEM_EQUALITY`,
`FILTERITEM_GREATEROREQUAL`,
`FILTERITEM_LESSOREQUAL`,
and `FILTERITEM_APPROX` searches all take an Attribute Value Assertion (AVA) structure, and return the entries for which the attribute in the entry matches the asserted value. The AVA structure is defined in Section 18.8.1.

fi_un.un_substrings: `FILTERITEM_SUBSTRING` matches, use the substring structure defined below, to specify which substrings to look for in the appropriate attribute.

```
typedef struct {
            AttributeType fi_sub_type;
            AV_Sequence   fi_sub_initial;
            AV_Sequence   fi_sub_any;
            AV_Sequence   fi_sub_final;
            char         *fi_sub_match;
        } Filter_Substrings;
```

The `AV_Sequence` structure is used to represent a string, and should be treated as essentially a linked list of `char *` parameters.

fi_sub_type: The attribute that you want to perform a substring search on.

fi_sub_initial: This contains a single attribute value, that must appear at the start of the string.

fi_sub_any: A set of values, which must appear in order in the middle of the string.

fi_sun_final: This contains a single attribute value, that must appear at the end of the string.

fi_sub_match: For DSA use only.

fi_ifp: For DSA use only.

The single filter items can now be linked into a `filter` structure to build more complex search definitions:-

```
typedef struct filter {
    char          flt_type;
#define FILTER_ITEM 1
#define FILTER_AND  2
#define FILTER_OR   3
#define FILTER_NOT  4
   struct filter * flt_next;
   union {
        struct filter_item   flt_un_item;
        struct filter      * flt_un_filter;
   }               flt_un;
}filter, *Filter;
```

flt_type: This defines whether the filter is a single item, or a more complex filter made up of one or more components.

flt_un.flt_un_item: If the filter represents a `filter_item`, then the item is placed here.

flt_un.flt_un_filter: `AND`, `OR` and `NOT` filters apply to a linked list of "children" filters. This element is a pointer to the head of that list.

flt_next: If the parent filter is an `AND`, `OR`. or `NOT` filter, then the component filters are linked using this field. NOTE that `NOT` filters should contain a list of one child only.

As an example, lets us consider a filter to represent a person whose name is either "Robbins" OR ("Steve" AND "Kille"). The structure would look something like:-

```
flt_type = FILTER_OR
        - The filter is an OR filter
flt_un_filter.flt_type = FILTER_ITEM
        - First component or the OR is an item
flt_un_filter.flt_un_item = filter_item (Robbins)
        - The item should match "robbins"
flt_un_filter.flt_next->flt_type = FILTER_AND
        - Second component or the OR is an AND filter
flt_un_filter.flt_next->flt_un_filter.flt_type = FILTER_ITEM
        - First component or the AND is an item
flt_un_filter.flt_next->flt_un_filter.flt_un_item =
                                        filter_item (Steve)
        - The item should match "Steve"
flt_un_filter.flt_next->flt_un_filter.flt_next->flt_type =
                                        FILTER_ITEM
        - Second component or the AND is an item
flt_un_filter.flt_next->flt_un_filter.flt_next->flt_un_item =
                                        filter_item (Kille)
        - The item should match "Kille"
```

18.11 Modification Operations

There are 4 operations available to modify the directory.

NOTE: with this version of QUIPU, modify operations are only allowed over DAP, attempts to modify over DSP will return referral errors.

18.11.1 Add

To add an entry to the DIT use

```
ds_addentry (arg, error)
    struct ds_addentry_arg      * arg;
    struct DSError              * error;
```

You will need to include *quipu/add.h* to use this procedure.

The argument you must supply is made up as follows:-

```
struct ds_addentry_arg {
    CommonArgs    ada_common;
    DN            ada_object;
    Attr_Sequence ada_entry;
};
```

`ada_common`: The common arguments as described in 18.2.1.

`ada_object`: The DN of the object you want to add.

`ada_entry`: The attributes you want to add for this entry. This must contain the RDN of the entry, and an "objectclass" attribute. Then, the other attribute MUST conform to the set of "optional" and "mandatory" attribute for that object class.

18.11.2 Remove

This is used to remove an entry from the DIT. Only leaf entries can be removed.

```
ds_removeentry (arg, error)
    struct ds_removeentry_arg   * arg;
    struct DSError              * error;
```

You will need to include *quipu/remove.h* to use this procedure.
The argument you must supply is made up as follows:-

```
struct ds_removeentry_arg {
    CommonArgs rma_common;
    DN         rma_object;
};
```

`rma_common`: The common arguments as described in 18.2.1.

`rma_object`: The DN of the object you want to remove.

18.11.3 Modify

`ModifyEntry` is used to modify the attributes of the specified entry. There are strong restrictions on this operation as required by X.500. Invalid operations result in errors, and as such none of the requested modifications are made. To modify an attribute you must first `remove` it then `add` a new attribute. You can not modify the RDN, you must use the `ModifyRDN` operation described in Section 18.11.4 to do this.

```
ds_modifyentry (arg, error)
    struct ds_modifyentry_arg   * arg;
    struct DSError              * error;
```

You will need to include *quipu/modify.h* to use this procedure.
The argument you must supply is made up as follows:-

```
struct ds_modifyentry_arg {
    CommonArgs         mea_common;
    DN                 mea_object;
    struct entrymod * mea_changes;
};
```

`mea_common`: The common arguments as described in 18.2.1.

`mea_object`: The DN of the object you want to modify.

`mea_changes`: A tree structure defining the changes you want to make to the entry.

```
struct entrymod {
    int               em_type;
#define EM_ADDATTRIBUTE        1
#define EM_REMOVEATTRIBUTE     2
#define EM_ADDVALUES           3
#define EM_REMOVEVALUES        4
    Attr_Sequence     em_what;
    struct entrymod * em_next;
};
```

`em_type`: The type of modification this is.

em_what: The attribute you want to add or remove. If the operation type is remove, then this structure should only contain an attribute type, and not a value.

em_next: A modify operation is built up with a series of small modifications. Hence this structure is a linked list. The operation is seen as one atomic operation.

18.11.4 ModifyRDN

To modify the distinguished attribute values of an entry, you MUST use this operation — `modifyEntry` can not be used.

```
ds_modifyrdn (arg, error)
    struct ds_modifyrdn_arg     * arg;
    struct DSError              * error;
```

You will need to include *quipu/modifyrdn.h* to use this procedure.
The arguments are:-

```
struct ds_modifyrdn_arg {
    CommonArgs mra_common;
    DN         mra_object;
    RDN        mra_newrdn;
    char       mra_deleterdn;
};
```

mra_common: The common arguments as described in 18.2.1.

mra_object: The DN of the object you want to modify the name of.

mra_newrdn: The new RDN.

mra_deleterdn: if `TRUE` then the old RDN will be deleted as an attribute, otherwise it will remain as a non-distinguished attribute.

18.12 Abandon

The abandon operation shown below is slightly different to the other DUA operations. It does not make much sense for a synchronous interface to handle abandon directly. In the next release, transparent use of abandon will be provided.

```
ds_abandon (arg, error)
    struct ds_abandon_arg       * arg;
    struct DSError              * error;
```

The argument structure contains a single parameter supplies the operation invocation id.

```
struct ds_abandon_arg {
    int aba_invokeid;
};
```

In the synchronous interface, and operation may be abandoned by the user entering a "control–c" (unless trapped by user code). In this case, the abandon will be handled by the lower layers of QUIPU, and the outstanding operation will return. It may have results if the abandon operation failed, or may indicate "abandon successful" via the DAP error structure!

18.13 Multiple Associations

The procedural interface described so far, has been based on the assumption of a connection to a single DSA. However, it is possible to make connection to multiple DSAs.

18.13.1 Multiple Binds

The bind and unbind routines needed to make multiple association are as follows:-

```
dap_bind (ad, arg, error, result, addr)
    int                         * ad;
    struct ds_bind_arg          *arg;
```

```
        struct ds_bind_arg          *result;
        struct ds_bind_error        *error;
        struct PSAPaddr             *addr;

    dap_unbind (ad)
        int                         ad;
```

The `arg`, `error` and `result` arguments are as defined in Section 18.5 for the routine `secure_ds_bind`.

The argument `ad` is the association descriptor of the association. It will be different of each association.

The `addr` argument in the address of the DSA you wish to contact.

18.13.2 Other DAP Operations

The other DAP operations (read, list, modify...), use the same format a already described, but with two extra integer parameters, thus

```
ds_read (arg, error, result)
```

becomes

```
dap_read (ad, id, arg, error, result)
```

where `ad` is the association descriptor returned by the bind, and `id` uniquely identifies the operation with respect to other operation on the same association.

Chapter 19

The Async DAP procedural interface

Originally, the procedural interface providing a representation of the X.500 DAP operations was only required to provide synchronous behaviour. It is likely that many useful application will be written which only require synchronous representation of operations: so that having bound to a DSA, operations are constructed in some way and then the routine representing the relevant operation is called and will block until an appropriate response has been received from the DSA.

However, in more sophisticated DUAs, it will be desirable to have asynchronous or non-blocking routines representing the various DAP operations, so that the DUA can construct and send operations, perform useful work whilst waiting for the operations to complete, and collect and use the responses to operations as they become available.

For the above reasons, a new set of procedure calls has been written as part of the *libdsap*(3n) library which provide a representation of the X.500 operations (including the bind operation) which can be used to program asynchronous performance of operations within a DUA. This chapter describes those routines and their intended use.

To use the async DAP interface described here it is necessary to include the file "`quipu/dap2.h`".

19.1 Procedure Model

19.1.1 Styles of Behaviour

The routines which it is desirable to have to represent X.500 DAP operations depends on the behaviour required within a DUA. A simple DUA may be more easily developed by expecting a synchronous behaviour from routines representing operations: DAP operations are invoked as routines which block until a response to the operation arrives and the DUA can then continue according to that response.

If, for whatever reasons, it is undesirable to block every time a DAP operation is requested, then a different behaviour is required from the routines representing operations: the operation request should be issued and the routine return, subsequently a routine should be available to accept a response to the request issued and process that response.

An additional twist is that there are actually three styles of behaviour which it is desirable to make available to DUA programs: a simple synchronous interface, an interruptible synchronous interface and an asynchronous interface.

The interruptible synchronous interface is the style of behaviour described in the previous section: if an operation is called synchronously and then interrupted (a Control-C to the user interface for instance) during performance of the operation, then an abandon DAP operation should automatically be invoked on the outstanding operation and appropriate responses gathered from the DSA before returning to the user. The interruptible synchronous style of behaviour is only available for the remote operations of the directory access context (read, add entry, ...) and not for the bind and unbind operations.

Mixing of styles of behaviour is deprecated, although there are no known problems, since an asynchronous operation request followed by a synchronous operation request may well get the response to either operation which means checking the responses to all synchronous requests in the same ways as the responses to asynchronous requests must be checked, thus losing most of the advantages that make the synchronous style worthwhile in the first place.

In order to enable the incorporation of all three styles a generalised interface has been written over which the interruptible synchronous interface described in Section 18.1 has been reimplemented. Thus, the routines in that previous section can be thought of as a specialised restriction of the interface described here.

The generalised interface also differs, in that it does not provide automatic decoding of results and errors into particular parameters, instead returning an indication parameter which may include decoded structures within it. This means that the implementation of the old style interface using the generalised routines is not as straightforward as the above comments might suggest.

Future revisions should take on board the separation of the provision of routines which perform DAP operations from the provision of routines which provide DUA services in the construction of DAP operations (e.g., getting passwords for a bind request is a high-level DUA task, parameterisation which allows the use of pre-allocated result and error return parameters is a lower-level DUA task) all the DAP operations should provide is the encoding of arguments, sending of operations and appropriate subsequent behaviour. Hopefully, future documentation will be produced which is structured so as to enable a clearer relationship between the different interfaces and behaviour styles provided for DAP operations.

Where each routine of the synchronous interface described in the previous chapter had a prefix of "`ds_`" (when using an implicit association identifier) or of "`dap_`" (when using an explicit association identifier), all the routines in the asynchronous interface have a prefix of "`Dap`". All operations in the asynchronous interface take an explicit association identifier.

For each of the DAP bind operation, the DAP unbind operation and the DAP remote operations (read, list, search, addentry, removeentry, modifyentry, modifyrdn and abandon) a request routine is provided. The request routine takes a parameter indicating the style of behaviour to be used (synchronous, or asynchronous for the bind and unbind operations requests; synchronous, interruptible or asynchronous for the DAP remote operations).

If the style indicated is synchronous, then the request routine will attempt to construct and send the appropriate operation and wait for a response. If the operation completes successfully (a result or an error is returned) then the routine returns `OK` otherwise, on failure, the routine returns `NOTOK`.

If the style indicated is interruptible, then the operation is sent and an interruptible wait initiated for a response from the DSA. If an interruption occurs (see a description of the routine `RoIntrRequest` in the rosap module for details of what constitutes an interruption and how it is indicated) then an abandon operation is sent for the outstanding operation and responses from the DSA are awaited for both the abandon operation and for the outstanding operation. Except in the case of an interruption the style of behaviour is exactly the same as the synchronous style.

If the style indicated is asynchronous, then the operation is sent and the routine returns. For the DAP bind and DAP unbind operations a retry routine is provided for each operation which when called will either complete the outstanding operation or indicate that it is not yet complete. For the DAP remote operations a single wait routine is available which will complete an outstanding operation, if there is one which can be completed, or indicate that there is no completable operation.

The value used to indicate the style of behaviour is borrowed from the "`rosap`" module:

`ROS_SYNC`: Simple synchronous behaviour: block until DSA responds.

`ROS_INTR`: Interruptible synchronous behaviour: block until DSA responds or an interruption occurs. (Untidily, the definition of the value `ROS_INTR` is currently contained in "`quipu/dap2.h`".)

`ROS_ASYNC` : Asynchronous behaviour: send operation, do not block.

19.1.2 Arguments

Arguments taken by async DAP request operations are, in general, a binding (or association) identifier which identifies the association on which to invoke the operation, an operation identifier which it is the responsibility of the DUA to generate, maintain and use as a reference (e.g., for abandoning a previously requested operation); a representation of the argument for the operation, an indication return parameter through which information on the outcome of the operation can be returned to the invoker of the routine, and of course the behaviour style indicator.

19.1.3 Indications

All of the routines representing X.500 operations in this procedural interface take an indication parameter, the type of which is a pointer to a pre-allocated `DAPindication` structure.

```
struct DAPindication {
    int     di_type;
#define DI_RESULT       2
#define DI_ERROR        3
#define DI_PREJECT      4
#define DI_ABORT        6
    union {
        struct DAPresult        di_un_result;
        struct DAPerror         di_un_error;
        struct DAPpreject       di_un_preject;
        struct DAPabort         di_un_abort;
    } di_un;
#define di_result di_un.di_un_result
#define di_error di_un.di_un_error
#define di_preject di_un.di_un_preject
#define di_abort di_un.di_un_abort
};
```

This structure is a discriminated union (tag element followed by a union). Depending on the routine called, style of behaviour requested and the value returned by the routine the contents of the indication should be readily retrievable.

For the bind and unbind routines the `DAPindication` structure is only updated on failure and will contain an abort indication.

For the DAP remote operations the `DAPindication` structure is used to indicate failure of the association, failure of the request, an error response to a request or a result response to a request.

When the `DAPindication` structure is used to indicate a failure of the association it will contain a `DAPabort` structure.

```
struct DAPabort {
    int     da_source;

    int     da_reason;

#define DA_SIZE 512
    int     da_cc;
    char    da_data[DA_SIZE];
}
```

da_source: the source of the abort, one of:

Value	Source
DA_USER	service-user (peer)
DA_PROVIDER	service-provider
DA_LOCAL	local DAPM

da_reason: The reason for aborting the association (if known).

da_data/da_cc: a diagnostic string from the provider.

When the `DAPindication` structure is used to indicate a rejection it will contain a `DAPpreject` structure.

```
struct DAPpreject {
    int            dp_id;

    int            dp_source;

    int            dp_reason;

#define DP_SIZE 512
    int      dp_cc;
    char     dp_data[DP_SIZE];
}
```

dp_id: the operation identifier of the operation rejected.

dp_source: the source of the abort, taking the same values as `da_source` in the `DAPabort` structure described above.

dp_reason: The reason for rejecting the operation (if known).

dp_data/dp_cc: a diagnostic string from the provider.

When the `DAPindication` structure is used to indicate a result in response to a previously issued remote operation request it will contain a `DAPresult` structure.

```
struct DAPresult {
    int                   dr_id;
    struct DSResult       dr_res;
}
```

`dr_id`: the operation identifier of the operation for which this is the result.

`dr_res`: the decoded result structure.

The operation identifier should be used on the receipt of an operation result to determine the operation the result applies to, and the result structure handled appropriately.

When the `DAPindication` structure is used to indicate a error in response to a previously issued remote operation request it will contain a `DAPerror` structure.

```
struct DAPerror {
    int                   de_id;
    struct DSError        de_err;
}
```

`de_id`: the operation identifier of the operation for which this is the error response.

`de_err`: the decoded error structure.

The operation identifier should be used on the receipt of an operation error to determine the operation the error applies to, and the error structure handled appropriately.

19.1.4 Return values

The values returned by the various routines in this interface and their meanings are more complicated than may be entirely reasonable. However, a quick overview goes as follows:

- for bind and unbind operations synchronous style and DAP remote operations in synchronous or interruptible style, the request routines may return `NOTOK` on failure and `OK` on success.

- for bind and unbind operations asynchronous style, the request routines may return `NOTOK` on failure, `DONE` on completion, and `CONNECTING_1` or `CONNECTING_2` when incomplete.
- for DAP remote operations asynchronous style, the request routines may return `NOTOK` on failure, `OK` on successful invocation.

19.2 Binding and Unbinding

19.2.1 Binding

To be able to send DAP remote operations to a DSA it is necessary to establish an association with the DSA using a DAP bind operation. This can be accomplished either synchronously or asynchronously using the routine `DapAsynBindReqAux`.

```
int       DapAsynBindReqAux (callingtitle, calledtitle,
                callingaddr, calledaddr, prequirements,
                srequirements, isn, settings, sf,
                bindarg, qos, dc, di, async)
AEI                        callingtitle;
AEI                        calledtitle;
struct PSAPaddr          * callingaddr;
struct PSAPaddr          * calledaddr;
int                        prequirements;
int                        srequirements;
long                       isn;
int                        settings;
struct SSAPref           * sf;
struct ds_bind_arg       * bindarg;
struct QOStype           * qos;
struct DAPconnect        * dc;
struct DAPindication     * di;
int                        async;
```

This routine will attempt to call a DSA at the called address and/or with the called title, to establish a directory access association using the directory bind argument provided.

Often, all that is required for many of these parameters is that they are initialised to appropriate values. The routine `DapAsynBindRequest` is provided which sets up sensible defaults for most of the parameters before calling the routine `DapAsynBindReqAux`.

```
int DapAsynBindRequest (calledaddr, bindarg, dc, di, async)
struct PSAPaddr         * calledaddr;
struct ds_bind_arg      * bindarg;
struct DAPconnect       * dc;
struct DAPindication    * di;
int                       async;
```

`calledaddr`: the address of the DSA to send the bind request to. The `PSAPaddr` structure is described in *Volume Two*), and more general discussion of addresses is given in *Volume One*).

`bindarg`: is a representation of the argument of the DAP bind operation. The `ds_bind_arg` structure is described in Section 18.1.

`dc`: is used to return bind response information when a response is received. The `DAPconnect` structure comprises connection information from the underlying association and a structure containing a representation of the bind response (bind result or bind error) if any.

`di`: is the indication parameter described above in Section 19.1.3.

`async`: can take one of two values: `ROS_ASYNC` or `ROS_SYNC`. If the value is `ROS_SYNC`, then the routine will await a response to the association request and attempt to decode the value returned in that response. If the value is `ROS_ASYNC`, then the routine may return without receiving an association response, which may be checked for later using the `DapAsynBindRetry` routine.

The `DAPconnect` structure is used to return the response to a DAP bind operation:

```
struct DAPconnect {
    int     dc_sd;
```

```
    int     dc_pctx_id;
    struct AcSAPconnect dc_connect;

    int     dc_result;
#define DC_RESULT       1
#define DC_ERROR        2
#define DC_REJECT       3

    union {
        struct ds_bind_arg        dc_bind_res;
        struct ds_bind_error      dc_bind_err;
    } dc_un;
};
```

`dc_sd`: the association identifier assigned to the bound association.

`dc_pctx_id`: the identifier of the directory access context in the negotiated list of presentation contexts.

`dc_connect`: the connect information for the underlying association.

`dc_result`: the type of response received.

`dc_bind_res`: the decoded bind result if the response is a result.

`dc_bind_err`: the decoded bind error if the response is an error.

For explanations of the other parameters to **DapAsynBindReqAux** see the description of the **AcAssocRequest** routine on page 30 of *Volume One*.

When the asynchronous style of behaviour is selected for a bind request, the **DapAsynBindRetry** can be used to attempt to complete the bind and process any response. In order to determine when to call **DapAsynBindRetry**, the methods for asynchronously establishing connections described in Section 4.2.3 on page 110 in *Volume Two* should be applied. Essentially, the values **CONNECTING_1** and **CONNECTING_2**, are used to determine whether the association establishment is currently blocked on reading or writing, and is used in constructing a call to **xselect** to determine if further progress can be made. If the call to **xselect** indicates that further work can be done then **DapAsynBindRetry** should be called.

```
int       DapAsynBindRetry (sd, do_next_nsap, dc, di)
int                        sd;
int                        do_next_nsap;
struct DAPconnect        * dc;
struct DAPindication     * di;
```

sd: the association descriptor returned in the `DAPconnect` structure after an asynchronous call to `DapAsynBindReqAux`.

do_next_nsap: is used to specify whether the association should be retried on the same nsap of the called address originally passed to `DapAsynBindReqAux` or whether to give up on the current nsap and go on to the next (if any). A value of zero will specify retrying on the same nsap, a non-zero value specifies retrying on the next nsap. This is somewhat messy as it forces the calling code to keep track of how long a particular nsap has been tried for and when to try another. If there are no more nsaps when the next nsap is requested, then the association attempt is deemed to have failed and an appropriate indication is generated.

dc: if the routine returns `DONE` then this structure will contain the connect information as described for `DapAsynBindReqAux` above.

di: if the routine returns `NOTOK` then this structure will contain the indication information.

19.2.2 Unbinding

When a DUA no longer needs to be bound to a DSA it can issue a DAP unbind operation to unbind from the DSA and end the association. This is achieved by using the `DapUnBindRequest` routine.

```
int       DapUnBindRequest (sd, secs, dr, di)
int                        sd;
int                        secs;
struct DAPrelease        * dr;
struct DAPindication     * di;
```

sd: the association identifier for the association from which the DUA wishes to unbind.

"`secs`: the number of seconds to spend trying to unbind.

`dr`: the result of unbinding when complete.

`di`: an indication.

If the unbind operation fails then the routine will return `NOTOK` and an appropriate indication.

If the unbind operation completes successfully within `secs` seconds then the routine will return `OK` and the `DAPrelease` structure will be filled out.

If no unbind response is received within `secs` seconds the routine will return a value of `DONE` and the unbind should be completed using the routine `DapUnBindRetry`.

```
int       DapUnBindRetry (sd, secs, dr, di)
int                       sd;
int                       secs;
struct DAPrelease       * dr;
struct DAPindication    * di;
```

Which takes exactly the same parameters and has similar behaviour to the routine `DapUnBindRequest` above.

19.3 DAP Remote Operations

For each DAP remote operation (read, abandon, add entry, ...) there is a routine in the asynchronous interface to request that operation. The `DapRead` routine will be described in detail, which taken in conjunction with the descriptions of arguments given in Section 18 should also provide adequate description of the routines `DapCompare`, `DapAbandon`, `DapList`, `DapSearch`, `DapAddEntry`, `DapRemoveEntry`, `DapModifyEntry`, and `DapModifyRDN`.

All these routines can be invoked with synchronous, interruptible or asynchronous styles of behaviour. In the synchronous and interruptible cases the call will not return until a response is received (in the interruptible style an interruption will cause an automatic sending of an abandon request for the operation requested by the routine and will await and return the response to that operation). In the asynchronous case the routine will return after the request has been sent and specific procedures for receiving responses must be undertaken using the `DapInitWaitRequest` routine.

19.3.1 Invoking requests

```
int        DapRead (ad, id, arg, di, asyn)
int                        ad;
int                        id;
struct ds_read_arg       * arg;
struct DAPindication     * di;
int                        asyn;
```

`ad`: the association identifier for the bound association over which to send the read operation.

`id`: the operation identifier to assign the the read operation.

`arg`: the read argument.

`di`: an indication parameter, only used if the request fails.

`asyn`: the behaviour style selector.

If `DapRead` is invoked with synchronous behaviour, then it will block until a response arrives, and hope that the response is for the operation just requested, although no guarantees are made that this is so.

If the interruptible style is selected then the call may be interrupted by some signals and an abandon operation for the operation requested is generated when this occurs, with the routine then awaiting both the response to the abandon operation and to the original operation before returning.

If the asynchronous style is selected, then the call issues the request and returns, the response should be expected in a subsequent call to the routine `DapInitWaitRequest`.

19.3.2 Receiving responses

```
int        DapInitWaitRequest (sd, secs, di)
int                        sd;
int                        secs;
struct DAPindication     * di;
```

`sd`: the association identifier.

`secs`: the number of seconds to spend waiting for a response; `OK` will produce a polling effect and `NOTOK` will produce a blocking effect.

`di`: the indication structure.

If the `DapInitWaitRequest` routine returns `OK` then there is a response contained in the indication parameter. If it returns `DONE` then there was no response available within the specified time.

19.4 Programming Comments

This interface has been written with expectations of the following sort of program structure in mind.

Through interaction with a user or from arguments or tailoring, a DUA will undertake to attempt one or more DAP bind operations to one or more DSAs.

An asynchronous call to `DapAsynBindReqAux` should be made using an appropriate bind argument for each DSA that the DUA wishes to bind to. If the association completes then operations can be constructed and invoked on the bound association identified by the value returned in the `dc_sd` fields of the `DAPconnect` structure `dc_sd`.

If the value returned by the `DapAsynBindRequest` procedure call is either `CONNECTING_1` or `CONNECTING_2` then this value should be recorded along with the value returned in the `dc_sd` field of the `dc` parameter, which identifies the association even though it is not yet complete. Operations MUST NOT be invoked using this identifier until the bound association is completely established.

At a later stage, for each outstanding association request, depending on whether the last value returned for the association attempt on that association identifier is `CONNECTING_1` or `CONNECTING_2`, the routine `xselect` should be used to check for writing or reading on the value returned by `PSelectMask` for the given association identifier.

If the call to `xselect` indicates that writing (or reading) is available then the routine `DapAsynBindRetry` should be called for that association identifier.

This should be repeated until the association attempt fails or is completed, at which point operations can be constructed and invoked over the bound association. This is done using the appropriate routine for the required operation.

If operations are invoked asynchronously, then a call to the procedure `DapInitWaitRequest` will need to check for responses to outstanding operations and to return such a response as an indication if there is one available. The DUA can then process responses to the operations it invoked.

When the DUA no longer needs to be bound to a DSA, the procedure `DapUnBindRequest` can be used to initiate the termination of a bound association, and if necessary, the `DapUnBindRetry` routine used to complete the binding.

Chapter 20

Caching in a DUA

This part of the manual is written for implementors who wish to perform caching in their DUA. This used to require the use of the the *libquipu*(3n) library. However the functionality has now been rolled into the *libdsap*(3n) library and the *libquipu*(3n) library removed.

The library provides some routines to manage the data returned by the DUA procedure calls described in Chapter 18.

20.1 The Entry Structure

The `Entry` structure is used by the QUIPU DSA to store the data in the local DIT. This structure can also be used by a DUA to cache information from a successful read operation, and so the structure is described. In fact, this is how *dish* maintain its cache.

The structure is shown below, many of the fields are not appropriate to a DUA, but are briefly described for completeness.

```
typedef struct entry {
    RDN             e_name;
    Attr_Sequence   e_attributes;
    InheritAttr     e_iattr;
    char            e_leaf;
    char            e_complete;
    int             e_data;
#define E_DATA_MASTER           1
```

```
#define E_TYPE_SLAVE              2
#define E_TYPE_CACHE_FROM_MASTER  3
#define E_TYPE_CONSTRUCTOR        4
    char                e_allchildrenpresent;
    struct acl        * e_acl;
    DN                  e_alias;
    struct dsa_info * e_dsainfo;
    char              * e_edbversion;
    AV_Sequence         e_oc;
    AV_Sequence         e_inherit;
    struct entry      * e_parent;
#ifdef TURBO_AVL
    Avlnode           * e_children;
#else
    struct entry      * e_sibling;
    struct entry      * e_child;
#endif
    time_t              e_age;
    char                e_lock;
    char                e_external;
                        /* 0 -> Quipu, 1 -> External */
    union  {
       struct {
       AV_Sequence un_master;
       AV_Sequence un_slave;
       } un_in;
       struct {
       int         un_reftype;
       AV_Sequence  un_reference;
       } un_out;
    } e_un;
#define e_master        e_un.un_in.un_master
#define e_slave         e_un.un_in.un_slave
#define e_reference     e_un.un_out.un_reference
#define e_reftype       e_un.un_out.un_reftype

    int         e_refcount;
} entry, *Entry;
```

`e_name`: The RDN of the entry, to find the DN, use the routine

```
DN get_copy_dn (entryptr)
Entry entryptr;
```

`e_attributes`: The attributes returned by the DSA.

`e_iattr`: pointer to attributes inherited into the entry. This will always be NULL in a DUA — all attributes whether or not inherited will be in the `e_attributes` field.

`e_leaf`: Set to `TRUE` if the entry is known to be a leaf.

`e_complete`: Set to `TRUE` if it is know that we have ALL the attributes.

`e_data`: This takes one of four values. in the DUA only two apply. `E_TYPE_CACHE_FROM_MASTER` implies the entry has data that has been read from the DSA. `E_TYPE_CONSTRUCTOR` is used when the entry has been "made" by the caching mechanism to fill in a missing part of the DIT on the way down to another entry. An entry of this type will contain no data except form the RDN of the entry.

`e_allchildrenpresent`: Set to `TRUE` when it is known that all the children are held. This is a DSA specific field.

`e_acl`: A pointer to the "Access Control List" attribute.

`e_alias`: Set to `TRUE` if the entry represents an alias. This is a DSA specific field.

`e_dsainfo`: pointers to the "EDBInfo" and "PresentationAddress" attribute of entries representing DSAs.

`e_edbversion`: This is a DSA specific field representing the EDB version number of the children.

`e_parent`: Pointer to the parent entry.

`e_sibling`: Pointer to other nodes at this level in the tree.

`e_child`: Pointer to the children.

`e_children`: Pointer to the children (if using AVL trees).

`e_age`: If the data type is `E_TYPE_CACHE_FROM_MASTER`, then the entry is time stamped. This enables the cache to be "timed out". NOTE in a DUA all entries are cached!

`e_lock`: This field has a dual purpose! In the DSA it is used to prevent an entry from being modified. In the DUA, if `FALSE`, then is suggests that in the field `e_attributes` there are only the attribute types for some attributes, this is a result of caching an results from an operation in which the entry info selection requested that attribute type only were returned (see Section 18.7.1).

`e_external`: Indicates type of reference — QUIPU or non-QUIPU.

`e_master`: Pointer to the "MasterDSA" attribute.

`e_slave`: Pointer to the "SlaveDSA" attribute.

`e_reftype`: Type of a reference to a non-QUIPU DSA.

`e_reference`: Reference to a non-QUIPU DSA.

`e_refcount`: Reference count used to make sure entry is not freed too early.

Using this structure a tree, mapping the DIT can be built up. The tree is rooted by the external `Entry` "`database_root`". This entry will NEVER have any attributes or siblings, ONLY children.

20.2 Caching Results

There are four routines to create a cache from results of DAP operations:-

To cache results returned from a `read` or `search` operation call

```
cache_entry (ptr, complete, vals)
EntryInfo      *ptr;
char            complete;
char            vals;
```

The `complete` parameter should be `TRUE` if all attribute were requested and returned.

The `vals` parameter should be `TRUE` if the attribute values were asked for.

The external `Entry` "`current_entry`" will be a pointer to the newly created entry.

To remove an entry from the cache use:-

```
delete_cache (adn)
DN              adn;
```

20.3 Finding Data in the Cache

The procedure

```
Entry local_find_entry (object,deref)
DN                      object;
char                    deref;
```

is used to find an entry in the cache. If the entry exists, it will be returned, otherwise `NULLENTRY` will be returned. `object` is the DN of the entry you want to find. If `deref` is `TRUE` then any aliases encountered in trying to find the entry will be de-referenced.

The cache is only maintained for the period specified in a `cache_timeout` external variable (this is initially set to six hours). If the cache expires then `local_find_entry` will return `NULLENTRY`. To prevent timeouts, the routine `local_find_entry_aux` will return the cached entry even if the timeout has expired.

20.4 Caching List Results

List results are cached slightly differently, as only the RDN is known.

```
cache_list (ptr, prob,dn,sizelimit)
struct subordinate *ptr;
int             prob;
DN              dn;
int             sizelimit;
```

The parameters are as follows:-

ptr: A pointer to the subordinates returned by the search.

prob: The `srr_limitproblem` field of the search results, see Section 18.10.

dn: The DN of the entry that has been listed.

ptr: The `svc_sizelimit` field of the common argument of the list argument (see Section 18.2.1).

To inspect the cache use

```
struct list_cache *find_list_cache (dn,sizelimit)
DN dn;
int sizelimit;
```

The `sizelimit` parameter is the sizelimit you would send in the list request.

This returns a `list_cache` structure, if the result is NULL, then the entry is not cached, or has less than "sizelimit" results.

```
struct list_cache {
        DN              list_dn;
        struct subordinate *list_subs;
        struct subordinate *list_sub_top;
        int             list_count;
        int             list_problem;
        struct list_cache *list_next;
};
```

The subordinates can be found in `list_sub_top`. The other field should be ignored, as they are for internal management only.

20.5 Changes

To conclude this part of the manual, a brief summary of the changes between this version and the QUIPU-6.0 version of the *libdsap*(3n) library is given.

Calls to the `x_decode()` routines are no longer required and should be removed. All the decoding is performed as the attributes come across the network.

The AttributeType structure is now just a pointer into the OID tables, thus "at_table" struct reference is no longer needed. So code of the form

```
at->at_table.oa_syntax
```

should be replaced with

```
at->oa_syntax
```

and

```
at->at_oid
```

replaced with

```
at->oa_at.ot_oid
```

Part V

Design

Chapter 21

Overview

21.1 Introduction

This part of the manual describes aspects of the design of QUIPU which are not needed to be known by the administrator or user of QUIPU. However, it documents important design decisions and protocol, which are of interest to understand how QUIPU works, and in some specific circumstances (e.g., solving interoperability problems). A summary of the main features of QUIPU is also given.

QUIPU fully implements both of the OSI Directory Protocols, and a number of extensions. The highlights of the QUIPU Directory Service Implementation are:

- Use of memory structures to provide fast access, without use of complex keying techniques.
- Activity scheduling within the DSA to allow for multiple accesses.
- General and flexible searching capabilities.
- A mechanism to provide non-local access control.
- A mechanism to provide external schema management.
- A sophisticated approach for management of distributed operations and replication.

The current implementation provides a DSA, and a procedural interface to the Directory Abstract Service and the associated Directory Access Protocol (DAP), which will enable other applications to use the Directory.

21.2 General Aims

To understand the rationale behind some of the decisions, it is useful to consider the original aims of the QUIPU project. These can then be mapped onto a number of more technical considerations:

- To produce an implementation which followed the emerging standards. This is an aim in itself.
- Flexibility, to enable the system to be used for experimentation and research into problems relating to directory services.
- To provide a vehicle for experimentation in the area of distribution and replication.
- To provide some level of real usage. This sort of work is useless if entirely confined to the laboratory. It is important that it is capable of use for some level of experimental service. However, it is not consciously designed to evolve into a full fledged product.

As the work has evolved, the following goals have emerged as additional to the original ones listed above:

- To provide a public domain the OSI Directory implementation as a part of the ISODE package.
- To provide integrated support for the ISODE Applications.
- To be used as a part of the initial pilot Directory Service in the UK Academic Community and in other pilots.

21.3 Technical Goals

The major goals of the QUIPU Directory Service are:

- Full support of the Directory Access Protocol and Directory System Protocols [CCITT88].
- Support of the majority of the service elements specified in the OSI Directory.
- Full interworking with other OSI Directory implementations.
- Very full searching and matching capabilities, beyond the minimum required by the OSI Directory.
- Provision of a system which has potential for very high distribution.
- Support of distributed operations in a manner which is in full conformance with respect to non-QUIPU systems, and provides additional functionality for QUIPU systems.

The following areas were not intended as goals in the initial system. Some discussion is given as to how these areas might be tackled in future versions.

- The QUIPU Directory is not intended for very large scale systems (i.e., Millions and tens of Millions of entries per DSA or hundreds of megabytes of data per DSA).
- Substantial data robustness is not required: there is no need to employ complex data backup techniques, such as replicated hardware.
- The security aspects of the OSI Directory were initially omitted, as not required by the general aims. At this point, there is no reason why this aspect should not be integrated.

21.4 Further QUIPU documents

The following documents are available, in addition to this manual:

- A paper on the original design, which is mainly of historical interest [SKill87].
- A paper presented at the 1988 IFIP 6.5 conference, which gives a general overview [SKill88a].
- A paper presented at Esprit Conference Week 1988, which describes the distributed operations [SKill88b].
- A paper presented to the Dutch UNIX User Group in November 1989, which describes how QUIPU DSAs navigate the DIT [PBark89].

These papers, except the first, are distributed online with QUIPU.

Chapter 22

General Design

22.1 Overview

This chapter describes general decisions. In particular, issues relating to use of the OSI Directory are covered, rather than system implementation decisions. However, the two are somewhat bound up. Attention is drawn to the protocol extensions defined in section 25.1. Note that this does *not* affect interactions with non-QUIPU DSAs (or DUAs). The following aspects of the OSI Directory are not handled in the current version of QUIPU:

- The protocol elements for support of directory use of authentication are handled in a conformant manner, but the associated services are not available to the end user.
- Search is always supported by multicasting. This does *not* affect the basic service offered to the user, but means that prohibition of chaining is not possible in all cases.
- Partial Outcome Qualifier is not supported for List.
- There are some aspects of distributed operation, where interaction with another conforming system would not be fully general. In particular, QUIPU might not be able to be configured with references to point at a complex configuration where not all sibling entries are held.

Otherwise, QUIPU is believed to conform to the standard.

22.2 Service Controls

QUIPU use of service controls conforms to the OSI Directory. Comments are made on those controls where QUIPU makes a choice with respect to some option given by the OSI Directory.

preferChaining: Chaining will be done.

chainingProhibited: Chaining will not be done. For some cases of the search operation, this means that the QUIPU Directory Service will not be able to provide the service, and will return a "chaining required" error.

localScope: The scope will be restricted to the DSA concerned (i.e., no chaining will be done).

dontUseCopy: If this is set, the master data will be used. This may have a significant impact on performance for operations on entries which are high up the tree and for the DSAs which master this information. These issues need study.

dontDereferenceAliases: Followed as per the OSI Directory.

priority: This is used to help control scheduling within the DSA. High priority tasks are dealt with before low priority tasks (Note: there are no checks here, so this is open to mis-use !)

timeLimit: Followed as per the OSI Directory.

sizeLimit: Followed as per the OSI Directory.

scopeOfReferral: The OSI Directory is followed, although QUIPU does not make use of this control.

Chapter 23

Distributed Operation

23.1 Overview

Distributed Operation is a major aspect of the QUIPU Directory Service Sadly, the OSI Directory specifications in this area are, in the author's opinion, rather unsatisfactory. Therefore, the QUIPU distributed operations are described in a slightly different manner. The concept of "Naming Context" is not used, and the significance of "Knowledge" is de-emphasised. The external view of this functionality is fully in line with the standard.

Some of the concepts defined in this chapter do *not* correspond to the ISO/CCITT terms, and so new terminology is introduced. Standard terminology is used in the standard way.

23.2 DSA/DUA Interaction Model

There are some interesting choices to be made between DSA Referral and Chaining. A DUA will start by contacting a local DSA, specifying that chaining is preferred (i.e., DSA referrals should not be passed back to the DUA). After that, the first DSA will proceed by use of DSA Referral, except for operations where this is not possible in the QUIPU framework (some cases of search). The advantages of this approach are:

- The DUA code can be kept to a minimum, as there is no need to handle referrals. This does mean that the DUA must always interact with the

Directory Service through a DSA which supports chaining (which might exclude some implementations).

- Always going thorough a local DSA allows a "per system" cache to be maintained in a coherent manner.
- The overhead of maintaining chained connections is not passed on too far.

Note that whilst the DUA procedural does not handle referrals transparently, they are defined in the service interface, so that an application can choose to handle the referrals directly if it wishes to do so.

23.3 Model of Data Distribution

This is a critical section of the design. It is essential to understand it before studying distributed operation.

23.3.1 Entry Data Blocks

For the root and every non-leaf vertex, there will be an *Entry Data Block* (EDB) which contains *all* information pertaining to the next level down in the DIT. Figure 23.1 gives and ASN.1 description of the Entry Data Block format.

```
<entrydatablock> ::= <type> <CRLF> <version> <CRLF> <data>
<type>           ::= "MASTER" | "SLAVE" |  "CACHE"
<version>        ::= <printablestring>
<data>           ::= <entry> | <entry> <CRLF> <data>
<entry>          ::= <rdn> <CRLF> <attributelist> <CRLF>
<attributelist>  ::= <attribute> | <attributelist> <CRLF> <attribute>
```

Figure 23.1: Entry Data Block Format

It should be noted and remembered that the Entry Data Block associated with an entry and described as "the Entry Data Block of the entry" contains the entries children (i.e., their attributes and RDNs) and not the attributes of the entry itself. This is *not* necessarily intuitive.

23.3.2 Masters and Slaves

Every Entry Data Block has *Master* and *Slave* copies. There will typically be only one master (although there may be multiple master copies, where data is maintained "out of band"). Slave copies are automatically replicated from a Master EDB. This may be direct or indirect. The full propagation path must be acyclic (loop free).

A DSA has either all or none of an Entry Data Block as a Master or Slave (viz: Entry Data Blocks are atomic). Any other DIT information it contains is treated as cached data. A DSA does not need to have any Master or Slave data.

DSAs are named, and represented in the DIT. One of the reasons for this, is to enable use of the Directory to identify the OSI location of DSAs. This OSI location can then be adjusted transparent to the logical mapping of the DIT onto DSAs. This can be seen as treating a DSA in the same manner as any other Application Entity. This simplifies the implementation, as there does not need to be specific storage of additional configuration information (knowledge).

An important piece of information stored in the entry of each DSA is the list of EDBs and how they are replicated. This information is the basis for automatic replication.

23.3.3 QUIPU Subordinate References

An entry has associated with it, attributes which indicate the DSAs which have Master or Slave Entry Data Blocks for the entry in question. These pointers are known as *QUIPU Subordinate References* (QSR). For every QSR, the DSA must have a Master or Slave copy of the EDB, as implied by the QSR. The converse it not true: DSAs may have copies of EDBs without there being QSRs. The DSAs with QSRs have the information *and* are prepared to answer public queries about the Entry Data Block in question. DSAs with EDBs (typically slave copies) and no QSRs usually have copies for performance or robustness reasons.

The QUIPU Subordinate Reference is similar to the standardised Non-Specific Subordinate References (NSSR). There are the following differences:

- QSRs admit to replication, and therefore there are Master QSRs and Slave QSRs.

- A QSR always points to all of the relevant information, whereas an NSSR may only point to a part of it and must be used in "and" conjunction with other NSSRs.

23.3.4 Access to the root EDB

There is no requirement for a given DSA to hold a copy of the root Entry Data Block. However, to be able to systematically process all queries, there must be direct or indirect access to the root Entry Data Block. Therefore, every DSA which does not have a copy of the root Entry Data Block must know the name and address of one or more DSA which either has a copy of the root Entry Data Block, or is closer (in terms of these references) to a DSA which has a copy. This approach is similar to, but not the same as, use of superior references defined in the standard.

23.4 Standard Knowledge References

In addition, to the QUIPU specific QSRs, an entry might also contain standard Knowledge References, as defined in the OSI Directory. This is used to point to data not contained in a QUIPU DSA. There are three types of reference defined in the standard:

Subordinate Reference: Pointer to an Entry

Cross Reference: From the QUIPU standpoint, the same as a subordinate reference

Non Specific Subordinate Reference: Pointer to multiple subordinates which must be queried for the next level down. QSRs are similar to this (see previous section).

In the first two cases, the entry in the Entry Data Block is considered to be "Knowledge only" (although other entry information may be cached). In the third case the entry will also have full information on itself. If any of these are present, there will be no QUIPU master or slave pointers. These three types of pointer are mutually exclusive[1].

[1] Thought(SEK) — does the standard let you have a subordinate reference plus a cached NSSR?

23.5 Navigation

Given this data model, a straightforward navigation algorithm can now be specified. The requirement is to locate the entry associated with a specified Distinguished Name. When the entry is arrived at, the operations will behave as proscribed.

The basis of the navigation strategy is that the first DSA (i.e., the one accessed by the DAP) does all of the hard work. Other DSAs, accessed by DSA Referral, either answer the question or return an error. This is important, as it is the basic strategy by which the system ensures completion of queries. There are times when the DSA may depart from this model, these are discussed in Section 23.8 and [PBark89].

First consider the behaviour of a DSA accessed by the Directory System Protocol (DSP):

1. Look up the Distinguished Name.

2. If the Distinguished Name is found, go to step 6.

3. If there is a local copy of the Entry Data Block of the parent, return a nameError. The "matched" parameter should be set to the Distinguished Name of the Entry Data Block (i.e., the level above the offered name). This is an authoritative NO.

4. Strip the lowest component off the Distinguished Name, and go to step 1. If there are no more components, go to step 5. This process checks for authoritative NO.

5. At this point, the name has not been found, and no relevant Entry Data Block has been found. This implies that the DSA does not hold the root Entry Data Block. Therefore the DSA should return a DSA Referral. The DSA Referral should be the list of DSAs (names and addresses) which are known to be closer to the root.

6. We now have an entry which matches some or all of the original Distinguished Name. Consider this entry.

7. If the entry contains an Alias attribute, dereference, and goto step 1. Note that if a referral is returned, that the appropriate parameters should be set to indicate all dereferences.

8. If the entry is the one specified in the query, return the answer to the query, or the appropriate (authoritative) error.

9. If the entry is of object class "QuipuNonLeafObject", return a Referral. This is simply a redirect to a DSA which can take the query at least one step further. The names for the DSA Referral should be taken from the master and slave QUIPU Subordinate References. Where the calling DSA is a non-QUIPU DSA, the Presentation address of the Master DSA must be looked up, and only this one returned.

10. If the entry contains a reference, the appropriate referral should be returned.

11. The query refers to an entry below the bottom of the DIT. An authoritative nameError can be returned.

Now consider the slightly more complex case of the initial DSA (doing the DSA Referral). Steps 1-4 are followed as above as above, to determine authoritative NO.

5. At this point, the name has not been found, and no relevant Entry Data Block has been found. This implies that the DSA does not hold the root Entry Data Block. The list of DSAs which are known to be closer to the root, are the starting point for the iterative query. Go to step 12.

6. We now have an entry which matches some or all of the original Distinguished Name. Consider this entry.

7. If the entry contains an Alias attribute, apply the relevant dereference to the original query, and go back to the start.

8. If the entry is the one specified in the query, return the answer to the query, or the appropriate (authoritative) error.

9. If the entry is of object class "QuipuNonLeafObject", this gives a list of QSRs to start the iterative query. Go to step 12.

10. If the entry contains a standard knowledge reference, then go to step 12. Note that for non-specific subordinate references, *all* of the references must be followed before giving up.

11. The query refers to an entry below the bottom of the DIT. An authoritative nameError can be returned,

12. Select one of the DSAs from the referral list. The order to try the DSAs is arbitrary. However, it is attempted to select ones with the topologically closest name first (e.g., a UK DSA will prefer to query another UK DSA before asking a French one). Try DSAs in turn until one gives an answer or you get bored. Consider the answer. Authoritative answers (yes or no) should be passed back to the DUA. If a Referral is received, recurse to step, watching carefully for loops.

It can be seen that this navigation is relying on data being distributed correctly, and DSAs other than the one doing the work behaving in a correct manner. Information provided in the referral should be used to ensure that the iteration is progressing, and thus detect livelock situations.

23.6 List

The Entry Data Block concept allows the list operation to fall out in a straightforward manner. Navigation to the Entry Data Block belonging to the name provided, will give access to the full result for the list operation.

Note that where cross/subordinate references are involved, it will be assumed that these are not alias entries (reasonable in practice). This will allow list to be performed in a single DSA in all cases.

23.7 Search

This section describes how the OSI Directory search is supported. This is one of the hardest parts of the implementation, and care must be taken. Note that the DAP argument in DSP is always that provided by the DUA[2]. This means that the work done by a given DSA must be in relation to the target object. With other operations, this is (fairly) straightforward, as the target object always references the base object of the operation. For searching, care

[2]Whilst this is in principle one of the key aspects of the way the DSP works, the recommendations for distributed operations violate this principle when dealing with aliases during search.

must be taken to correctly verify whether the base object has been reached. This is done by use of the operation progress information, setting the name resolution phase to completed.

The search operation functions by searching the part of the tree implied by the "subset" specification. Rather than returning all of the information, the queried DSA will apply the associated filter to the entries in question, and return the filtered result, along with appropriate continuation pointers. This should minimise network traffic.

The case of "subset = baseObject" is handled by navigating to the Entry Data Block of the object's parent, and applying the given filter. If the entry is a cross reference or subordinate reference, the reference should be followed (using the same query).

The case of "subset = oneLevel" is handled by navigating to the object's own Entry Data Block. There the filter is applied to each of its children. If any of the children are alias entries, the alias should be de-referenced, and a baseObject search applied to the new entry. For each child which is a cross references or subordinate references, the references should be followed, setting the target object to be the child.

The case of "subset = wholeSubtree" is handled by navigating to the object's own Entry Data Block. There, the filter is applied to the object and to each of its children. If any of the children are alias entries, the alias should be de-referenced, and a wholeSubtree search applied to the new entry. For each child which has QUIPU children (determined by the prescence of a masterDSA attribute), the search should be applied to the master or one of the slave DSAs, with target object set to the child. For each child which is a cross reference or subordinate reference, the references should be followed, setting the target object to the child. For each child which has non-specific subordinate references, the search should be applied to *all* of the referenced DSAs, with the target object set to the child.

There are three procedures for operating:

1. Everything handled by the first DSA, with other DSAs returning a mixture of results and partial outcome qualifiers.

2. Proceed by referral until a DSA is reached which has a copy of the base object. Then this DSA proceeds by referral, and returns the full result to the first DSA. This is how The current version of QUIPU works.

It has the advantage of often accumulating search results in a local environment, and so is selectable (possibly in a complex manner).

3. Proceed by referral until a DSA is reached which has a copy of the base object. Then proceed entirely by chaining. This is not done.

There is potential for looping in this procedure. This will be detected and broken by noting loops in the DSA trace information. This takes account of the fact that some distribution will allow for a query to re-enter the same DSA a number of times.

The Search and list operations make use of the "partial results" functionality to return information if a time or size limit is reached. Thus, setting a low size limit will allow a user to easily examine sample information (either by list or search).

23.8 Selecting a DSA

In QUIPU-5.0 the chain/refer choice was very ad hoc. For a DSA, the best choice is usually to give a referral, except where this will not work. To make this calculation, it is necessary to determine if two DSAs can communicate directly. This is done by deriving from the presentation address of a DSA, a list of connected networks. This can then be used to determine if a pair of DSAs can communicate directly, and is the basis for the chaining/referral choice. This will need the extension of the network address, to allow encoding of private networks other than the three well known ones.

There is an analogous problem when a DSA needs to access a DSA which cannot be accessed directly. Each DSA which does not have full connectivity, will have an attribute which indicates network/DSA pairs. This indicates a DSA which may be used (bilateral agreement) to access a given network by application relay.

The following sections discuss briefly how these choices are made, [PBark89] describes the process more fully.

23.8.1 DSA Quality

Replication gives a choice of DSAs to direct a given query to. The following criteria are relevant:

- Use an existing association if possible
- Prefer a QUIPU DSA
- Prefer a reliable DSA
- Prefer a "local" DSA

The first two are straightforward. A local DSA can be selected using the following...

1. Prefer to use networks in the order specified by ts_communities. This will encourage access over a local ether/preferred net.
2. Pick a DSA with a close name (e.g., prefer one in the same country)
3. Pick a DSA in the same DMD. This would need to add a DMD attribute to the DSA entry (encoded as DN) [Not yet implemented].

Picking a reliable DSA is achieved using the following information

```
DSAInfo ::= SEQUENCE  {
     dsa DistinguishedName,
     lastAttempt UTCTime,
     lastSuccess UTCTime OPTIONAL,
     failures-since-last-success INTEGER }
```

Thus a DSA will be able to check if it knows about the DSAs in question, and can make a choice based on past results. This should operate dynamically, without operator interference. Information on DSAs not contacted for a given period should be expunged. It is hoped to store this as an attribute of a DSA in future versions of QUIPU.

23.8.2 Unavailable DSAs

In the case where a DSA is unavailable, and chaining is preferred, a reference will be returned by a QUIPU DSA. A QUIPU DUA, which knows it is talking to a QUIPU DSA can rely on this behaviour, and simply use the referral as a diagnostic to the user. It is hoped that the next version of the standard will add an obvious extra parameter.

23.8.3 Operating When DSAs are not Fully Interconnected

Whilst global interconnection of all application entities is an OSI ideal, it will not be achievable in the short or medium term. Application relaying by DSAs will be needed to achieve full directory connectivity.

In general, it is not desirable for DSAs to proceed by chaining — it wastes unnecessary application level resources. Later, there may be policy reasons to prefer chaining, but these are ignored for now. The internal structure of network addresses allows a DSA to determine if two DSAs can communicate directly.

23.9 The External View of QUIPU

To a non-QUIPU system, QUIPU will appear to work exactly according to the standard.

When a QUIPU DSA interacts with another DSA, it will look up its object classes (and probably other information). This will allow it to determine if the other DSA is a QUIPU DSA. When interacting with another QUIPU DSA, the following deviations from the standard are possible. These are primarily concerned with the introduction of replication:

- Presentation Address might be omitted from Access Point (always present in the current QUIPU version).
- Cross References and Subordinate References have multiple values (although QUIPU will probably never send these to itself).
- Multiple values of Non Specific Subordinate Reference are assumed to have OR conjunction (i.e., they are really QSRs).
- Use of QUIPU Access control as described in Section 24.3.5

When a QUIPU DSA returns references which are derived from reference attributes, it will return them as specified. If it returns information derived from QUIPU internal pointers, it will return a non-specific subordinate reference. If the DSA being communicated with is not a QUIPU DSA, it will return only a reference to the a selected DSA (as replication is not admitted within the protocol).

23.10 Cached Data

Cached data is not mentioned in the basic algorithm. However, the algorithm can utilise cached data in some circumstances. This is because of the manner in which identification of copy data has been introduced in the final stage of the OSI Directory specification. Cached data may be used whenever this is not prohibited by the service controls. The standard does not clearly define what "copy" data is. In general, QUIPU treats master and slave data as authoritative. Both slave and cached data are returned to the user as "copy" data. For this reason, the distinction between slave and copy data can only be internal to QUIPU.

There is no time to live or age information in the OSI Directory Protocols. Care must be taken when caching, that spurious information is not passed around indefinitely between DSAs.

When QUIPU holds cached data, it will notes how long it has had it, and will "time out" the data after a tailorable period.

This section is open ended. The exact approaches to caching, and determining suitable timeout values will be the subject of experiment.

The important thing about managing cached data is to handle timeouts sensibly. Data cached from a cache may have an indeterminate age. It is important that this data is given a relatively short timeout, to prevent it being circulated indefinitely amongst a set of DSAs. However, *if* the data can be verified by usage (e.g., correctly connecting to a DSA verifies a presentation address), it should then be treated as if cached from a master/slave. Non-verified data should be treated in the same manner as user data, which is described in the next section. Data cached from master/slave information should be given a longer timeout. Data is discarded, rather than refreshed automatically. A timeout of some number of days seems appropriate for most data.

23.11 Configuration and Slave Update

A given DSA will have copies of zero or more Entry Data Blocks. A DSA may either be a master for a given Entry Data Block, or a slave. If there are multiple master copies, it is assumed that these are kept coherent by some out of band mechanism. For example, one of them is the "real" master, and the others are updated by file transfer when modifications occur. This discussion will proceed for the single master case.

There are three distinct types of DSA:

1. A DSA with a master copy of the root Entry Data Block.
2. A DSA with a slave copy of the root Entry Data Block.
3. A DSA with no copy of the root Entry Data Block.

As noted in Section 23.3, DSAs of type 2 and 3 will have pointer(s) to a DSA which is "closer" to the master copy of the root Entry Data Block. Specifying this hierarchical distribution, as opposed to requiring direct access to the master (as in earlier versions of the OSI Directory) means that there can be many copies of information which needs to be highly replicated, without excessive redundant copying across the Wide Area Network. This will be particularly important for the root Entry Data Block.

DSAs of type 2 will only need the pointer information for initial startup or recovery after catastrophic corruption. When the slave copy of the root Entry Data Block has been obtained for the first time, this will supersede the pointers. DSAs of type 3 will usually use cached information in preference to these pointers, and will only need the pointers if cached information is (or appears to be) invalid.

The only information which a DSA has to obtain locally at initial boot time, other than the DSA pointers, is its own name. All other information may be obtained from the Directory. Beyond this, the Directory Service manages its own configuration. There is little point in having a Directory Service providing general high speed access to global information, and then requiring an additional system (knowledge) to deal with its own configuration.

Associated with the DSAs entry in the DIT is a specification of the entries for which the DSA is a master, and for which it is a slave. A DSA will be able to derive the location of an Entry Data Block for which it is master from this information. Thus at initial boot, a DSA will utilise its initial DSA pointers to read its own entry. The location of master Entry Data Blocks will be derivable from their name, and so their existence can then be verified by the DSA in question. A DSA which is a master for the root Entry Data Block will have no pointers. However, it can go straight to the master root

Entry Data Block, read the information about itself, and proceed as for other DSAs.

It is believed that for early pilots, a high level of copying configuration data will be desirable to achieve robustness. The root and national EDBs will be very highly replicated, even though QUIPU can operate with a rather low level of replication.

23.12 DSA Naming

23.12.1 Choice of Names to Prevent Loops

Care must be taken to prevent the situation where the location of a DSA is only known through itself (and other more complex variants). A simple rule for naming DSAs will ensure that this cannot happen. The master DSA for a given entry (i.e., the DSA controlling the Entry Data Block of containing the entry's children) should have its name in the Entry Data Block of the entry's parent or at a level higher in the DIT. For example, the master DSA of

```
Country=UK, Org=University College London, OU=Computer Science
```

which contains information on entries below Computer Science, may be labelled

```
Country=UK, Org=University College London, DSA=Three Toed Sloth
```

or

```
Country=France, DSA=Capybara
```

It may not be labelled

```
Country=UK, Org=University College London, OU=Computer Science,
            DSA=Alpaca
```

or

```
Country=France, Org=Inria, DSA=Llama
```

A little more flexibility could be allowed; However, this rule is simple, it prevents deadlock, and allows for reasonable labelling practices. The restriction may be relaxed somewhat, when the concept of Directory Management Domains is introduced more formally.

Chapter 24

Access Control and Authentication

24.1 Models

24.1.1 Access Control

QUIPU uses access control lists to represent access rights; the subjects are DUAs (represented by DNs) and the objects are entries in the DIT, attributes of entries, and the child-of relation for each entry (all represented by DNs, and, in the case of attributes, the attribute type as well).

Furthermore, objects can be *containers* that hold other objects. To access an object within an container, it is necessary to have access rights both to the object and the container that holds it. In particular, entries in the DIT are containers. The attributes of the entry and the list of its children are contained within the entry. The children themselves are *not* contained within the parent.

The possible levels of access are as follows: *none*, *detect*, *compare*, *read*, *add* and *write*.

24.1.2 Security Domains

The DIT is a global database, maintained by many separate organisations. It is possible for the manager of DSA to change its interpretation of the access control rules, in order to fraudulently obtain access to information held by other DSAs.

As a result, data in a DSA may need to be protected from the managers of other DSAs, as well as from users. This means that special checks must be performed on DSP operations that come from untrusted DSAs.

In order to decide to trust a DSA, it is necessary to be able to authenticate it and to know that it should be trusted. The current version of QUIPU can do neither of these, and so assumes that *all* DSAs are untrusted.

24.2 Representation in the DIT

24.2.1 Simple Authentication

DUAs which use simple authentication have their password stored in the *userPassword* attribute of their entry.

24.2.2 Protected Simple Authentication

QUIPU represents both passwords (the K_A's) and authenticators by the ASN.1 type *ProtectedPassword* shown if Figure 24.1.

When this structure represents a password, *algorithm* indicates which one-way function is being used and *password* is the (unencrypted) password. The other fields are not supplied.

When this structure represents an authenticator, *password* is the hash value $f_1(K_A, t_1, q_1)$, where t_1 is *time1*, q_1 is *random1* and K_A is the password. *algorithm* is not supplied.

A relation $\geq$ can be defined on the set of passwords and authenticators as follows: If a is a password, b is an authenticator, and a matches b, then $a \geq b$. (Also, $a \geq a$). It can be shown that this relation is a partial ordering of the set, justifying our use of the symbol "$\geq$". We could have defined an attribute syntax for which the above relation corresponded to "greater than or equal" without violating any of the implied semantics of X.500. However, the DAP *compare* operation cannot be used to test "greater than or equal", only "equal", and we wanted an easy way to ask the Directory to check this relation. To achieve this, we made the relation correspond to "equals" for the *ProtectedPassword* attribute syntax.

```
ProtectedPassword ::=
    SEQUENCE {
    algorithm [0] AlgorithmIdentifier OPTIONAL,
    salt [1] SET {
        time1 [0] UTCTime OPTIONAL,
        time2 [1] UTCTime OPTIONAL,
        random1 [2] BIT STRING OPTIONAL,
        random2 [3] BIT STRING OPTIONAL}
        OPTIONAL,
    password [2] OCTET STRING}
```

Figure 24.1: ProtectedPassword

24.2.3 Access Control Lists

The access control list for an entry is held in that entry's *accessControlList* attribute.

24.2.4 Security Policies

The *entrySecurityPolicy* attribute of an entry is used to indicate the amount of care that should be taken to preserve the integrity and confidentiality of that entry. While the *accessControlList* indicates who should have access to the entry, the *entrySecurityPolicy* indicates which steps should be taken to prevent unauthorized access.

The *dsaDefaultSecurityPolicy* attribute of a DSA indicates which precautions the DSA will take when dealing with entries that do not have an *entrySecurityPolicy* attribute.

The *dsaPermittedSecurityPolicy* attribute of a DSA indicates which security policies the DSA is prepared to enforce. A DSA may not support a security policy either because it lacks the necessary software or because its manager wishes to forbid use of that policy.

The security policy attribute syntax is not yet implemented.

24.2.5 Labels

Security labels are one means of enforcing rule-based access control (sometimes referred to as mandatory access control). QUIPU does not provide mandatory access control in any form.

24.3 Distributed Operations

24.3.1 (Protected) Simple Authentication

If the responding DSA holds the initiator's entry, then it may check the password directly. Otherwise, the responder will formulate a DSP compare operation to check it. If the result is *true*, the bind will be accepted.

This approach can only be used to check a DAP bind; If it were used in DSP, there would be a danger of livelock. Hence, QUIPU will not attempt to verify the password in a DSP bind.

24.3.2 Strong Authentication

In strong authentication, it is necessary to build a certification path for the originator that will be believed by the recipient. From the standpoint of the protocol, the originator builds the certification path and sends it to the recipient. However, the originator does not know for certain which CA's the recipient trusts, and so cannot be sure of constructing a valid path.

QUIPU solves this problem by treating the presented certification path as a hint. The recipient tries to build an acceptable certification path out of the components of the presented path and the certificates it has cached.

24.3.3 Restricting Read Access

To maintain the confidentiality of data, results from read, search etc. must not be passed to another DSA unless that DSA has rights to them. The current version of QUIPU solves this problem in the manner described below.

When a DUA requests private data via an untrusted DSA, QUIPU checks whether any of the requested information can be seen by the DUA but not by the DSA. If it can, the security error "inappropriate authentication" is sent to the untrusted DSA. (Otherwise, the data can be safely sent over DSP).

QUIPU DSAs will interpret this error if it is sent by another QUIPU DSA to mean that they are not trusted, and will pass a referral back to the DUA. (Other DSAs may behave differently — X.500 ought to be clarified in this area).

The DUA will chase the referral, and repeat its query directly to the DSA holding the data. The DSA will then give the DUA the results.

24.3.4 Restricting Write Access

To maintain the integrity of the data, requests to modify data over DSP must not be accepted unless they are signed or can be performed by anyone.

QUIPU solves this problem as in the read access case, by returning a security error. When strong authentication is added to QUIPU, modify requests will be accepted over DSP provided that they are signed by the DUA.

As an optimisation, QUIPU DSAs never chain modify requests unless they are signed. (The operation will probably be rejected, so it's quicker to give a referral to the DUA immediately, rather than trying to chain).

24.3.5 Caching

DSAs try to improve performance by caching the results of DSP operations. Caching interferes with access control in two ways:

1. The cached data may be incomplete.

   The DUA that requested it may not have had rights to all the data, or the DSA that held the data may not have been prepared to send all of it over DSP.

2. Caching can prevent access control.

   The original data may have been subject to access controls which the holder of the cache is unaware of.

QUIPU has a special solution that only works between QUIPU DSAs (using the QUIPU DSP application context): The ACL is sent along with the data across DSP.

QUIPU DSAs will only cache data if they have the ACL to go with it. This ensures that the same access controls will be applied to a cache as to the master copy. It also enables a DSA to tell if the cache is complete. The required information is checked against the ACL in the cache; if any of it is not publicly readable, then it will not have been returned over DSP and the cache cannot be used.

24.3.6 Replicated Data

The mechanism originally envisaged for replicating data was "reliable ROS" — Remote operations carried by X.400(88). DSAs could use the security mechanisms in X.400(88) to provide both integrity and confidentiality for the EDB updates. This would make the slave updates the only place in QUIPU where cryptographic techniques are used to provide confidentiality, as opposed to integrity. The provision of confidentiality here is justified in view of the sensitivity of entire entry data blocks; many of them will contain user's passwords, for example.

However, X.400(88) is not yet widespread, and hence cannot be used as the primary means of replicating data. In the interim, the *getEDB* mechanism is used. This does provides neither confidentiality nor integrity, and is not even subject to access control (as DSP is unauthenticated).

Chapter 25

Replicating Updates

25.1 Basic Update Approach

QUIPU supports a simple automatic update mechanism. This allows for copying of Entry Data Blocks, but with a simple check to ensure that only new information is copied. Slave copies are obtained by use of a new remote operation. The argument to the operation is the name of the Entry, and the version number of the copy of the Entry Data Block held locally. A FULL copy of the Entry Data Block is returned if this version is out of date. In the DSAs entry, there is a list of Entry Data Blocks for which the DSA has master or slave copies, and the DSA which it gets updates from. For each Entry Data Block, there is the list of DSAs which pull the Entry Data Block, and for slave copies, which DSA the update should come from. It is assumed that this operation will be invoked sufficiently often for it to be acceptable to consider the slave data as "official". For the type of usage being considered, this probably means several times per day. Within QUIPU, this operation is in a new protocol (QUIPU DSP), which will also contains the DSP ASEs. The operation is specified in Figure 25.1.

```
GetEntryDataBlock ABSTRACT−OPERATION
        ARGUMENT GetEntryDataBlockArgument
        RESULT GetEntryDataBlockResult
        ERRORS {NameError,ServiceError,SecurityError}

getEntryDataBlock GetEntryDataBlock ::= 10

                                -- will make this an OBJECT IDENTIFER
                                -- when ISODE can support this form
                                -- of operation code

GetEntryDataBlockArgument ::= SET {
        entry [0] DistinguishedName,
        sendIfMoreRecentThan [1] EDBVersion OPTIONAL
                        -- if omitted, just return version held
                        -- To force send, specify old version
}

GetEntryDataBlockResult ::= SEQUENCE {
                versionHeld [0] EDBVersion
                [1] EntryDataBlock OPTIONAL
        }
}


EDBVersion ::= UTCTime
```

Figure 25.1: EDB Access Operation

Note that a DSA receiving a GetEDB operation, should check the associated EDBInfo, to ensure that the DSA in question is allowed to pull a copy of this EDB.

The operation may be used to determine which version of the EDB is currently master. This might be used when a query with dontUseCopy arrives, in order to determine whether slave information is accurate. This would be a big performance win for search and list operations, due to potential reduction in information transferred.

Chapter 26

Implementation Choices

26.1 DSA Structure

Whilst the operation of a QUIPU DSA is fast, its startup procedure is not, due to reading all of its data from a text file on disk. This long startup means that applications must be able to use multiple DSAs, to prevent lockout whilst the local DSA starts up. Also, the process structure of DSAs must be static. To provide a system with reasonable availability, particularly in view of the system's ability to perform extravagant searching, the basic DSA must be able to handle multiple calls. For this reason, apart from conformance issues, the DSA will be inherently asynchronous, and will need to have its own internal scheduling. Initially, this can be simple minded. However, we are providing a framework for a system which is very sophisticated in this area.

The basic approach of the DSA is to have two (conceptually) co-routined modules, which are interfaced by in-core C structures. The first module is the DSA engine, which resolves inbound queries either by looking them up in its in-core data structures or by generating further queries. The second module is the protocol engine. This is responsible for opening and closing calls, and for mapping between OPDUs on the network and C structures to be handled by the DSA engine. The interface provided to the DSA is largely independent of DAP vs DSP.

26.1.1 Memory Structures

There are a number of structures which are of particular importance. They are summarised here:

- The Entry is as the basic component of the in-core tree, which is linked upwards and downwards between parents and children. The tree always starts at the root, even if there is no information beyond the RDN present. Where an entry corresponds to the base of an Entry Data Block, the parameters of the Entry Data Block are present. Siblings are linked in a chain. Each entry is represented by a "C" structure, which contains:
  - Information on the linkage (hierarchical, and between siblings).
  - How Entry Data Blocks are managed.
  - How the "special" attributes (ACL, Schema, Alias, Password, DSA location info) are held.
- There is a structure associated with each connection. This is used to represent actual and desired connections. These structures are linked into a list, and are the key point for the protocol module. They indicate a list of operations and tasks associated with each connection. When the DSA engine needs a connection, it will see if one is already open to the DSA in question. If it is not, a connection structure will be created, which the protocol engine will act on in due course. Similarly, the protocol engine will close down unneeded connections, possibly after some (intentional) delay.
- There is a Task structure associated with each query which arrives. This holds the *full* state of the task, so the the DSA can switch between tasks at intervals (typically when a network connection blocks). This points to the list of operations which have been generated by the task. This is the key structure for the DSA Engine.
- There is an Operation structure, associated with each pending operation. These structures are in mesh structure, arranged both by Task and Connection.

Multi-level priorities are associated with tasks and operations, which are used by both engines to control scheduling. This is done in QUIPU on two bases:

- The user specified priority
- The progress of the operation (long searches are downgraded in priority).

26.1.2 Malloc

The above is optimised by careful use of malloc. A purpose built malloc is used, this knows about the memory intensive DSA, and tries to ensure that data accessed at similar times during the DSA operation, will be stored in the same page of core memory. This has the effect that the number of page faults generated is significantly reduced (early results indicate a twenty percent improvement over the standard malloc supplied with SunOS4).

26.1.3 Disk Structures

All of the data for a given QUIPU DSA will be contained under a single (UNIX) directory. There will be a directory for each RDN, where the DSA in question has an Entry Data Block (which may be master, slave, or cached). The name of directory being that of the QUIPU text encoded RDN. Thus there will be a UNIX directory structure which corresponds to the portion of the DIT held in the DSA. There will be file in each RDN directory called "EDB", which has the authoritative version of the data, and one called "EDB.bak", which has the previous version. This might be extended to provide more comprehensive backup.

When the system is booted, the following will occur:

1. Read tailoring information.
2. Look for sequence of Entry Data Blocks implied by the DSAs name. These will usually be cached. for later reuse. If not, the default addresses must be used.
3. With the DSAs own info available, read in the other master and slave Entry Data Blocks.

4. Read in other Entry Data Blocks, checking for consistency.

26.2 OSI Choices

ROS (1988) and the implied protocols (ACSE and Presentation) will be used. Other combinations (e.g., TP0 over TCP or TP4 over CLNS may be used by bilateral agreement between DUA and DSA or DSA and DSA).

To ensure full connectivity of the QUIPU Directory Service, one of the following conditions must be met:

1. Support of transport class 0 over international X.25(80) (This condition will be changed to support of CONS with access to the international X.25 subnetwork, when such a statement is realistic).

2. Access to a DSA which will perform application relaying in line with the procedures of Section 23.8.3. This will need a bilateral agreement. It is hoped to establish "well known" DSAs which can serve this function for the following well known networks:

   - Janet
   - The DARPA/NSF Internet

Part VI

Appendices

Appendix A

The QUIPU Pilot DIT

Table A.1 show the number of registered QUIPU DSAs at the top two levels in the DIT as of February $5^{\underline{th}}$ 1991. The number of entries is an estimation of the number of entries held by the DSAs in those countires. As not all DSA were contactable when the figures were calculated, this is almost certainly an underestimate. As the table shows, the QUIPU pilot is now active in 13 countries, with 177 DSAs, holding data about 370 organisations. Bearing in mind these figures only take account of DSAs at the top two levels of the DIT, and several DSAs are known to be represented lower down in the tree, we can estimate that there are about 200 DSAs in all holding close to 400 000 entries.

As an example of the typical spread of data, Table A.2 shows the organisations participating in the United Kingdom pilot. The size figure is only an estimate, and based on the information held by the relative DSAs.

Table A.1: Countries involved in QUIPU Pilot

Country	DSAs	Organisations	Entries
Australia	14	15	16536
Canada	10	10	21300
Denmark	1	1	16
Finland	10	13	10316
Germany	9	85	9026
Iceland	1	1	185
Netherland	2	69	2132
Norway	3	40	3714
Spain	2	3	141
Switzerland	5	9	4554
Sweden	6	17	3035
United Kingdom	38	33	67345
United States	66	74	167577
Total	177	370	296440

Table A.2: The c=GB Node of the QUIPU Pilot

Organisation	DSAs	Size
l=Nottingham X-Tel Services Ltd	3	3714
Aston University	1	935
Bell-Northern Research Northern Telecom	2	453
Bradford University	1	169
Brunel University	1	6900
Cambridge University	3	10647
Concurrent Computer Corporation	1	2660
Data General	1	24
Edinburgh University	2	3063
GID Ltd STC	1	19330
Glasgow University	1	2
Heriot-Watt University	1	20
Hewlett-Packard	1	7
Imperial College	1	2218
Joint Network Team	1	131
Manchester Computing Centre	1	73
Nottingham University	2	3647
Oxford University	1	89
Rutherford Appleton Laboratory	1	1710
Salford University Business Services Ltd	1	4
Salford University	1	4
Sussex University	1	1054
The University of Birmingham	1	1573
University College London	2	5101
University of Bath	1	1565
University of Exeter	1	26
University of London Computer Centre	1	148
University of Stirling	1	275
University of Strathclyde	1	1718
University of Warwick	1	85

Appendix B

BNF used QUIPU

This appendix gives a BNF definitions used by QUIPU.

Figure B.1 shows the bnf used for the oidtables described in Section 14.11.

Figure B.2 shows the bnf used to define Attributes and hence Distinguished Names, this is discussed in Section 3.2.3. The syntaxes used to represent attribute values are discussed fully in Chapter 10, and so are not repeated here.

Figure B.3 shows the bnf definition of the EDB files discussed in Section 14.1.1.

```
-- This specification is in BNF
-- Comments start with two dashes

<a> ::= any of the 52 upper and lower case IA5 letters
<d> ::= any IA5 digit 0-9
<k> ::= any of th 52 upper and lower case IA5 letters, IA5 digits,
            and "-" (hyphen)
<p> ::= any IA5 character in ASN.1 PrintableSting
<CRLF> ::= IA5 Newline
<letterstring> ::= <a> | <a> <letterstring>
<numericstring> ::= <d> | <d> <numericstring>
<keystring> ::= <k> | <k> <keystring>
<printablestring> ::= <p> | <p> <printablestring>
-- The first notation is to specify Object Identifiers.
-- These have the basic (BNF):

<numericoid> ::= <numericstring> | <numericstring> "." <numericoid>


-- We define a table which gives a mapping of generic OIDs to strings:
-- and a possible abbreviated form of the name

<oidkeytable> ::= <oidkeyentry> | <oidkeyentry> <CRLF> <oidkeytable>
<oidkeyentry> ::= <abbrstring> ":" <oidnumber>
<abbrstring>  ::= <keystring> | <keystring> "," <keystring>
<oidnumber>   ::= <numericoid> | <numericoid> "," <aliasoid>
<aliasoid>    ::= <numericoid>

-- For example:
-- UCL :0.3.2342.19200149
-- country,c :2.5.4.6

-- Tables of this form will be read in by every QUIPU DSA, to give it a set
-- of OID strings forms, and abbreviations

-- A general BNF of oid is now given:

<oid> ::= <keystring> "." <numericoid> | <keystring> | <numericoid>
```

```
-- For example, "UCL.5" gives OID 5 as allocated by UCL, "C" gives the
-- standard country oid.

- We define a table for attribute values:

<attrTable> ::= <attrEntry> | <attrEntry> <CRLF> <attrTable>
<attrEntry> ::= <oidkeyentry> ":" <attrEncoding>
<attrEncoding> ::= "ObjectClass" | "DN" | "CaseIgnoreString" |
             "CaseExactString" | "PrintableString" | "CountryString" |
             "Guide" | "PostalAddress" | "TelephoneNumber" |
             "telexNumber" | "TelexTerminalIdentifier" |
             "FacsimileTelephoneNumber" | "NumbericString" |
             "DestinationString" | "PresentationAddress" |
             "OID" | "OctetString" | "IA5String" | "Photo" |
             "Mailbox" | "UTCTime" | "DeliveryMethod"
             "Integer" | "Boolean" | "Password"
             "ACL" | "Schema" | "Update" |
             "Audio" | "CaseIgnoreList" | "VisibleString" |
             "Certificate" | "CerfiticatePair" | "CertificateList" |
             "ProtectedPassword" | "AccessPoint" | "Edbinfo" |
             "InheritedAttribute" | "NRSInformation" |
             <attrASN>

<attrASN> ::= <keystring>
     -- defined, but unknown syntax - treated as "ASN"

-- For example:
-- description: attributetype.13: CASEIGNORESTRING

-- Finally we define an object class table:

<ocTable> ::= <ocEntry> | <ocEntry> <CRLF> <ocTable>
<ocEntry> ::= <ocData> | <ocMacro>
<ocData>  ::= <oidkeyentry> ":" <strList> ":" <strList> ":" <strList>
--                        :(hierarchy):(must contain):(may contain)
<ocMacro>    :: = <keystring> "=" <strList>

<strList> ::= <keystring> | <keystring> "," <strList> |
```

```
-- Note the final "|" to permit the null string

-- For example;
-- localeAttributeSet = facsimileNumber,isdnAddress,telephoneNumber, ...
-- country: objectclass.2 : top: countryname: description,searchguide
-- organisationalPerson: person: : localeAttributeSet,OU,title,...
```

Figure B.1: BNF used for oidtables

```
<attribute-value> ::= <NumericValue>
                | <ASNValue>
                        -- For unknown syntaxes & photos
                | <DNValue>
                | <OIDValue>
                        -- OID & ObjectClass
                | <PSAPvalue>
                | <StringValue>
                        -- Used for:
                        --      CaseIgnoreString, CaseExactString,
                        --      PrintableString, CountryString,
                        --      OctetString, IA5String, Password
                        --      TelephoneNumber
                | <GuideValue>
                | <TelexNumberValue>
                | <TeletexTerminalIdentifierValue>
                | <FacsimileTelephoneNumberValue>
                | <PostalAddressValue>
                | <MailboxValue>
                | <UTCTimeValue>
                | <DeliveryValue>
                | <IntegerValue>
                | <BooleanValue>
                | <ACLValue>
                | <SchemaValue>
                | <EdbInfoValue>
                | <otherValues>

<otherValues>   ::= ANY -- can be user defined.


<namevalue>     ::= <namelist> ["#"]
<namelist>      ::= <name> | <namelist> "$" <name>

<attribute>     ::= <oid> "=" <avlist>
<avlist>        ::= <attribute-value> | <avlist> "&" <attribute-value>
<rdn>           ::= <attribute> | <attribute> "%" <rdn>
<name>          ::= <rdn> | <rdn> "@" <name>
                        -- most significant first
```

−− *An example name might be:*
−− *"c=GB @ o=UCL % locality=London @ ou=CS"*
−− *Finally, this can be assembled together to give a format for a textual*
−− *representation of an Entry Data Block.*

Figure B.2: BNF used in names

<entrydatablock>	::=	<type> <CRLF> <version> <CRLF> <data>
<type>	::=	`"MASTER"` | `"SLAVE"` | `"CACHE"`
<version>	::=	<printablestring>
<data>	::=	<entry> | <entry> <CRLF> <data>
<entry>	::=	<rdn> <CRLF> <attributelist> <CRLF>
<attributelist>	::=	<attribute> | <attributelist> <CRLF> <attribute>

Figure B.3: BNF used in EDB files

Appendix C

The QUIPU Naming Architecture

```
QuipuNameDefinitions DEFINITIONS ::=

BEGIN

IMPORTS
ABSTRACT−OPERATION, ABSTRACT−ERROR
   FROM AbstractServiceNotation
         {joint−iso−ccitt mhs−motis(6) asdc(2) modules(0) notation(1)}

NameError, ServiceError, SecurityError
   FROM DirctoryAbstractService
         {joint−iso−ccitt ds(5) modules(1) directoryAbstractService(2)}

DistinguishedName, RelativeDistinguishedName, Attribute,
ATTRIBUTE, ATTRIBUTE−SYNTAX, OBJECT−CLASS
   FROM InformationFramework
         {joint−iso−ccitt ds(5) modules(1) informationFramework(1)}

caseIgnoreStringSyntax
   FROM SelectedAttributeTypes
         {joint−iso−ccitt ds(5) modules(1) selectedAttributeTypes(5)}
```

```
TreeStructure, ACL, EDBInfo, MasterDSA, SlaveDSA, DSA, QuipuVersion,
InheritedAttribute, SubordinateReference, Crossreference,
NonSpecificSubordinateReference,
QuipuDSA, QuipuObject, QuipuNonLeafObject, ExternalNonLeafObject,
   FROM QuipuDirectoryDefinitions;

quipu OBJECT IDENTIFIER ::= {ccitt data(9) pss(2342) ucl(19200300) quipu(99)}
                                    -- interim QUIPU OID

attributeType OBJECT IDENTIFIER ::= {quipu attributeType(1)}

attributeSyntax OBJECT IDENTIFIER ::= {quipu attributeSyntax(2)}

objectClass OBJECT IDENTIFIER ::= {quipu objectClass(3)}

treeStructure TreeStructure ::= {attributeType 1}

acl ACL ::= {attributeType 2}

eDBInfo EDBInfo ::= {attributeType 3}

masterDSA MasterDSA ::= {attributeType 4}

slaveDSA SlaveDSA ::= {attributeType 5}

control DSAControl ::= {attributeType 15}

quipuVersion QuipuVersion ::= {attributeType 16}

ProtectedPassword ::=
      SEQUENCE
      {
            algorithm [0] AlgorithmIdentifier OPTIONAL,
            salt [1] SET {
                  time1 [0] UTCTime OPTIONAL,
                  time2 [1] UTCTime OPTIONAL,
                  random1 [2] BIT STRING OPTIONAL,
                  random2 [3] BIT STRING OPTIONAL
                  }
```

```
                OPTIONAL,
            password [2] OCTET STRING
        }


protectedPassword ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX    ProtectedPassword
        ::= {attributeType 17}

SecurityPolicy ::= ANY
              −− to be defined

entrySecurityPolicy ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   SecurityPolicy ::= {attributeType 18}

dsaDefaultSecurityPolicy ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   SecurityPolicy ::= {attributeType 19}

dsaPermittedSecurityPolicy ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   SecurityPolicy ::= {attributeType 20}

inheritedAttribute InheritedAttribute ::= {attributeType 21}

execVector ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   PrintableString ::= {attributeType 22}

relayDSA ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   distinguishedNameSyntax ::= {attributeType 23}

audio ATTRIBUTE
   WITH ATTRIBUTE−SYNTAX
   OCTECT STRING ::= {attributeType 24}

subordinateReference SubordinateReference ::= {attributeType 25}
```

```
crossReference Crossreference ::= {attributeType 26}

nonSpecificSubordinateReference NonSpecificSubordinateReference
                        ::= {attributeType 27}

listenAddress ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX
   PresentationAddress ::= {attributeType 28}

cachedEDB ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX
   distinguishedNameSyntax ::= {attributeType 29}


quipuDSA QuipuDSA ::= {objectClass 1}

quipuObject QuipuObject ::= {objectClass 2}

quipuNonLeafObject QuipuNonLeafObject ::= {objectClass 6}

externalNonLeafObject ExternalNonLeafObject ::= {objectClass 9}


quipuSecurityUser OBJECT-CLASS
   SUBCLASS OF quipuObject
   MUST CONTAIN {protectedPassword}
       ::= {objectClass 7}

iSODEApplicationEntity OBJECT-CLASS
   SUBCLASS OF applicationEntity
   MUST CONTAIN {execVector}
       ::= {objectclass 8}

friendlyCountryName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
  caseIgnoreStringSyntax
         ::= {attributeType 8}
                  -- example "UK", "United Kingdom" etc.
```

140

```
friendlyCountry OBJECT-CLASS
  SUBCLASS OF country, quipuObject
  MUST CONTAIN { friendlyCountryName }
          ::= {objectClass 3}


END
```

Figure C.1: The QUIPU Naming Architecture

Appendix D

ASN.1 Summary

This Appendix summarises the ASN.1 used (or planned for use) in QUIPU, and incorporates the system naming aspects.

Figure D.1: Summary of ASN.1

Appendix E

Attribute Matching

An important part of QUIPU is the ability to search for specified attributes. This appendix describes the attribute you can search for and the type of search that is valid, using the standard tables.

E.1 Approximate matches

The attributes shown in Figure E.1 take a "string" value, and perform approximate matching if asked. Substring matches are allowed. Exact matches are case independent.

Table E.2 shows the attributes for which matches are case sensitive.

E.2 Exact matches only

Table E.3 shows attributes for which matches are for equality only. The syntax of the value is a formatted string — the syntax of which is given shown.

Attribute	Abbreviation
knowledgeInformation	
commonName	cn
surname	
serialNumber	
countryName	c
localityName	l
stateOrProvinceName	st
streetAddress	
organizationName	o
organizationalUnitName	ou
title	
description	
businessCategory	
postalAddress	
postalCode	
postOfficeBox	
physicalDeliveryOfficeName	
userid	uid
textEncodedORaddress	
rfc822Mailbox	mail
info	
favouriteDrink	drink
roomNumber	
userClass	
host	
documentIdentifier	
documentTitle	
documentVersion	
documentLocation	
friendlyCountryName	co
orAddressComponent	

Table E.1: Case Independent Attributes

Attribute
telephoneNumber
telexNumber
teletexTerminalIdentifier
facsimileTelephoneNumber
iSDNAddress
registeredAddress
destinationIndicator
preferredDeliveryMethod

Table E.2: Case Sensitive Attributes

Attribute	Syntax
objectClass	objectclass
x121Address	numericstring
supportedApplicationContext	oid
aliasedObjectName	dn
member	dn
owner	dn
roleOccupant	dn
seeAlso	dn
manager	dn
documentAuthor	dn
masterDSA	dn
slaveDSA	dn
filestore	dn

Table E.3: Exact Match Only Attributes

Bibliography

[BKern78] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language. Software Series*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[CCITT88] The Directory — Overview of Concepts, Models, and Service. International Telegraph and Telephone Consultative Committee, December, 1988. Recommendation X.500.

[FSiro88] F. Sirovich and M. Antonellini. The THORN X.500 Distributed Directory Environment. In *Esprit '88: Putting the Technology to Use*, pages 1711–1720, North Holland, November, 1988.

[ISO88] Information Processing Systems — Open Systems Interconnection — The Directory — Overview of Concepts, Models, and Service. International Organization for Standardization and International Electrotechnical Committee, December, 1988. International Standard 9594-1.

[IUCN82] The IUCN Mammal Red Data Book. International Union for Conservation of Nature and Natural Resources, 1982. Unwin Brothers Limited, The Gresham Press, Woking, Surrey, UK. ISBN 2–88032–600–1.

[JPost81] Jon B. Postel. *Transmission Control Protocol.* Request for Comments 793, DDN Network Information Center, SRI International, September, 1981. See also MIL-STD 1778.

[MRose86] Marshall T. Rose and Dwight E. Cass. OSI Transport Services on top of the TCP. *Computer Networks and ISDN Systems*, 12(3), 1986. Also available as NRTC Technical Paper #700.

[MRose90a] Marshall T. Rose. *PSI White Pages Pilot Project: User's Handbook*. 1990. ISODE documentation set.

[MRose90b] Marshall T. Rose. *PSI White Pages Project: Administrator's Guide*. 1990. ISODE documentation set.

[MRose90c] Marshall T. Rose. *The Open Book: A Practical Perspective on Open Systems Interconnection*. Prentice-hall, 1990. ISBN 0–13–643016–3.

[PBark89] P. Barker and C.J. Robbins. Directory navigation in the quipu x.500 system. In *UNIX & Connectivity*, pages 235–244, National UNIX Systems User Group The Netherlands, November, 1989.

[SKill87] Stephen E. Kille. INCA Directory Services. In *Esprit '87: Achievements and Impact*, pages 1472–1476, North Holland, September, 1987.

[SKill88a] S.E. Kille. The QUIPU directory service. In *IFIP WG 6.5 Conference on Message Handling Systems and Distributed Applications*, pages 173–186, North Holland Publishing, October, 1988.

[SKill88b] S.E. Kille and C.J. Robbins. Distributed operations in QUIPU. In *Esprit Conference Week*, North Holland Publishing, November, 1988.

[SKill89a] Stephen E. Kille. *A string encoding of Presentation Address*. Research Note RN/89/14, Department of Computer Science, University College London, February, 1989.

[SKill89b] Stephen E. Kille. *The Design of QUIPU*. Research Note RN/89/19, Department of Computer Science, University College London, March, 1989.

[SKill89c] Stephen E. Kille. *The THORN X.500 Naming Architecture*. THORN Project Internal Report UCL-45, revision 6.1, University College London, January, 1989.

[SKill90] S.E. Kille. *Using the OSI Directory to achieve User Friendly Naming.* Research Note RN/90/29, Department of Computer Science, University College London, February, 1990.

Index