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

- The new *pepsy*(1) compiler which will replace *pepy* and *posy* programs in a future release of the ISODE.

- 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 "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 utilising 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. 7.0 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 the ISODE Manual.
- The `na_type` and `na_subnet` fields of the network address structure described in the ISODE Manual 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.
- An interworking problem in the presentation layer has been fixed in a backwards compatible manner. This should not affect interoperability in this release. The backwards compatibility will be removed after this release.

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.

Part II

File Transfer, Access and Management

Chapter 2

User Library

The *libftam*(3n) library implements the filestore-independent parts of the International Standard of the OSI file service, FTAM (which stands for File Transfer, Access and Management). Currently supported are: the no-recovery FTAM Quality-of-Service; the transfer, access, management, and transfer and management service classes; the kernel, read, write, access, limited file management, enhanced file management, grouping, and fadu-locking functional units; and, the kernel, storage, security, and private attribute groups.

Unlike most OSI services, FTAM distinguishes between the *initiator* of an FTAM association and the *responder*. However, as with most models of OSI services, an asynchronous environment is assumed. That is, the service provider may generate events for the service user without the latter triggering the actions which led to the event. For example, in a synchronous environment, an indication that data has arrived usually occurs only when the service user asks the service provider to read data; in an asynchronous environment, the service provider may interrupt the service user at any time to announce the arrival of data.

The `ftam` module in this release initially uses a synchronous interface; however once the connection is established, an asynchronous interface may be selected.

All of the routines in the *libftam*(3n) library are integer-valued. They return the manifest constant `OK` on success, or `NOTOK` otherwise.

2.1 Warning

Please read the following important message.

> **NOTE:** Readers of this chapter should have an intimate understanding of FTAM. It is not the intent of this chapter to present a tutorial on these services, so novice users will suffer greatly if they choose to read further.
> As previous versions of this software included an implementation of DIS FTAM, and the release contains an IS implementation, users of the *libftam*(3n) library are urged to re-read this chapter.

2.2 Constants

There are several important constants, described below:

2.2.1 FTAM Quality-of-Service

Value	FTAM-QoS
FQOS_NORECOVERY	no recovery
FQOS_CLASS1	class 1 recovery
FQOS_CLASS2	class 2 recovery
FQOS_CLASS3	class 3 recovery

Currently, only the no-recovery FTAM-QoS is supported.

2.2.2 Service Classes

Value	Service Class
FCLASS_TRANSFER	transfer
FCLASS_ACCESS	access
FCLASS_MANAGE	management
FCLASS_TM	transfer and management
FCLASS_UNCONS	unconstrained

Currently, all service classes other than the unconstrained class are supported.

2.2.3 Functional Units

Value	Functional Unit
FUNIT_READ	read
FUNIT_WRITE	write
FUNIT_ACCESS	file access
FUNIT_LIMITED	limited file management
FUNIT_ENHANCED	enhanced file management
FUNIT_GROUPING	grouping
FUNIT_FADULOCK	fadu locking
FUNIT_RECOVERY	recovery
FUNIT_RESTART	restart data transfer

Currently, all functional units other than the recovery and restart units are supported. Further, the grouping unit *must* be specified.

2.2.4 Attribute Groups

Value	Attribute Group
FATTR_STORAGE	storage
FATTR_SECURITY	security
FATTR_PRIVATE	private

Currently, all attribute groups other than the private group are supported.

2.2.5 State Results

Value	State Result
FSTATE_SUCCESS	success
FSTATE_FAILURE	failure

2.2.6 Action Results

Value	Action Result
FACTION_SUCCESS	success
FACTION_TRANS	transient error
FACTION_PERM	permanent error

2.3 Data-Structures

There are several important data structures, described below:

2.3.1 Contents Type

An FTAM document type is represented by the `FTAMcontent` structure.

```
struct FTAMcontent {
    OID     fc_dtn;

    int     fc_id;
    int     fc_result;
};
```

This elements of this structure are:

`fc_dtn`: the document type name, represented as an object identifier (consult Section 5.4.6 of *Volume One*);

`fc_id`: the presentation context identifier (consult Section 2.3.1 of *Volume Two*) associated with the document type; and,

`fc_result`: the status indicator for the presentation context (codes are listed in Table 2.1 of *Volume Two*).

A list of FTAM documents types is represented by the `FTAMcontentlist` structure.

```
struct FTAMcontentlist {
    int     fc_ncontent;

#define NFCONT  (NPCTX - 2)
    struct FTAMcontent fc_contents[NFCONT];
};
```

This elements of this structure are:

`fc_contents/fc_ncontent`: the contents list (and the number of contents in the list).

A limitation of this implementation is that a fixed number of document types may be supported, as denoted by the manifest constant `NFCONT`.