- 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

Application Services

Chapter 2

Association Control

The *libacsap*(3n) library implements the Association Control Service (ACS). Logically, one views an application to consist of several *application service elements* (ASE's). Although these ASEs cooperate, each performs a different function for the application. The Association Control Service Element (ACSE) is concerned with the task of "starting" and "stopping" the network for the application. That is, an application uses the ACSE to establish a connection, termed an *association*, between two users. The association *binds* the two users, which are referred to as the *initiator* and the *responder*. This association, once established is used by other ASEs (e.g., the remote operations service element). Later, the ACSE is called upon to release the association, either gracefully (perhaps with some negotiation), or abruptly (with possible loss of information).

Like most models of OSI services, the underlying assumption is one of an asynchronous environment: the service provider may generate events for the service user without the latter entity 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 `acsap` module in this release presents a synchronous interface. However once the association is established, an asynchronous interface may be selected. The `acsap` module itself is naive as to the interface being used: the particular application service element which is responsible for transferring data will manage the association once the ACSE has established it.

All of the routines in the *libacsap*(3n) library are integer-valued. They return the manifest constant `OK` on success, or `NOTOK` otherwise. In some circumstances, failures are fatal and cause (or are caused by) the association being released. Section 2.7 describes how such failures may be distinguished.

2.1 An Important Note

In the current release there are several ways to establish an association. This is due to recent changes in the OSI application layer, and a desire to remain compatible with previous work. Users are strongly encouraged to use the new facilities described herein, as the older facilities will eventually be removed.

Here are the new facilities available:

Desired Service	Association Primitives	Section(s)
Remote Operations (complete discipline) [ISO88e, CCI88c]	A-ASSOCIATE A-RELEASE A-ABORT	2.2 3.3.1
Remote Operations (complete discipline with reliable transfer) [ISO88e, CCI88c]	RT-OPEN RT-CLOSE	4.1 3.3.1
Reliable Transfer [ISO88c, CCI88a]	RT-OPEN RT-CLOSE	4.1

Here are the old facilities available:

Desired Service	Association Primitives	Section
Remote Operations (basic discipline) [ECM85]	RO-BEGIN RO-END	A.1
Remote Operations (advanced discipline with reliable transfer) [CCI84b]	X.410 OPEN X.410 CLOSE	A.2 3.3.1
Reliable Transfer [CCI84b]	X.410 OPEN X.410 CLOSE	A.2

In short, depending on whether a Reliable Transfer Service Element (RTSE) is desired, for the purposes of association control, all new applications should use either the library discussed below, or the reliable transfer routines discussed in Section 4.1 on page 87.

2.2 Associations

There are three aspects of association management: *association establishment*, *association release*, and, *association abort*. Each of these are now described in turn.

2.2.1 Association Establishment

The *libacsap*(3n) library distinguishes between the user which started an association, the *initiator*, and the user which was subsequently bound to the association, the *responder*. We sometimes term these two entities the *client* and the *server*, respectively.

Addresses

Addresses for the association control service entity consist of two parts: a presentation address (as discussed in Section 2.2 on page 16 of *Volume Two*), and application-entity information. Internally, the `AEInfo` is used to represent this notion:

```
typedef struct AEInfo {
    PE      aei_ap_title;
    PE      aei_ae_qualifier;

    int     aei_ap_id;
    int     aei_ae_id;

    int     aei_flags;
#define AEI_NULL        0x00
#define AEI_AP_ID       0x01
#define AEI_AE_ID       0x02
}       AEInfo, *AEI;
#define NULLAEI         ((AEI) 0)
```

This structure contains several elements:

`aei_ap_title`: the application-process title;

`aei_ae_qualifier`: the application-entity qualifier;

`aei_ap_id`: the application-process invocation identifier;

`aei_ae_id`: the application-entity invocation identifier; and,

`aei_flags`: flags, indicating which, if any, of the invocation-identifiers are present.

The routing `sprintaei` can be used to return a null-terminated string describing the application-entity.

```
char   *sprintaei (aei)
AEI     aei;
```

Finally, the routine `oid2aei` converts object identifiers (discussed in Section 5.4.6 on page 135) and converts them to application-entity information. This is used by the stub-directory service:

```
AEI     oid2aei (oid)
OID     oid;
```

In Figure 2.1, an example of how one constructs the address for the File Transfer, Access and Management (FTAM) service on host `RemoteHost` is presented. The key routine is `_str2aei`: