- 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

Underlying Services

Chapter 2

Presentation Services

The *libpsap2*(3n) library implements the presentation service. The kernel subset of the ISO specification is currently implemented. That is, the library supports whatever session requirements the user wishes to employ, negotiates presentation contexts on connection establishments, and utilizes abstract transfer notations to transmit data structures in a machine-independent manner.

As with most models of OSI services, the underlying assumption is one of a symmetric, asynchronous environment. That is, although peers exist at a given layer, one does not necessary view a peer as either a client or a server. Further, 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 `psap2` module in this release initially uses a client/server paradigm to start communications. Once the connection is established, a symmetric view is taken. In addition, initially the interface is synchronous; however once the connection is established, an asynchronous interface may be selected.

All of the routines in the *libpsap2*(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 the OSI presentation and session services. 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.

2.2 Addresses

Addresses at the presentation service access point are represented by the `PSAPaddr` structure.

```
struct PSAPaddr {
    struct SSAPaddr pa_addr;

#define PSSIZE  64
    int     pa_selectlen;
    char    pa_selector[PSSIZE];
};
#define NULLPA  ((struct PSAPaddr *) 0)
```

This structure contains two elements:

`pa_addr`: the session address, as described in Section 3.2 on page 42; and,

`pa_selector/pa_selectlen`: the presentation selector.

In Figure 2.1, an example of how one constructs the PSAP address for the LOOP provider on host `RemoteHost` is presented. The routine `is2paddr` takes a host and service, and then consults the *isoentities*(5) file described in Chapter 7 of *Volume One* to construct a presentation address.

```
struct PSAPaddr *is2paddr (host, service, is)
char   *host,
       *service;
struct isoservent *is;
```

```
#include <isode/psap2.h>
#include <isode/isoservent.h>

...

register struct PSAPaddr *pa;
register struct isoservent *is;

...

if ((is = getisoserventbyname ("loop", "psap")) == NULL)
    error ("psap/loop");

/* RemoteHost is the host we're interested in,
   e.g., "gremlin.nrtc.northrop.com" */

if ((pa = is2paddr (RemoteHost, NULLCP, is)) == NULLPA)
    error ("address translation failed");

...
```

Figure 2.1: Constructing the PSAP address for the LOOP provider

2.2.1 Calling Address

Certain users of the presentation service (e.g., the reliable transfer service) need to know the name of the local host when they initiate a connection. The routine `PLocalHostName` has been provided for this reason.

```
char   *PLocalHostName ()
```

2.3 Connection Establishment

Until the connection has been fully established, the implementation distinguishes between clients and servers, which are more properly referred to as *initiators* and *responders*, to use OSI terminology.

2.3.1 Connection Negotiation

From the user's perspective, there are several parameters which are negotiated by the presentation providers during connection establishment. Suggestions as to the values of some of these parameters are made by the user.

Session Parameters

Consult Section 3.3.1 for a list of session parameters which are negotiated during connection establishment.

Presentation Contexts

A *presentation context* is a binding between an abstract syntax notation and an abstract transfer notation on a presentation connection, and is denoted by an integer value termed the context identifier. The abstract syntax notation describes, to the users of the presentation service, the data structures being exchanged; the abstract transfer notation describes, to the providers of the presentation services, the method for encoding those data structures in a machine-independent fashion. Hence the abstract syntax notation is negotiated by the users of the presentation service, while the abstract transfer notation is negotiated by the providers of that service.

When a connection is established, the initiator suggests zero or more presentation contexts, specifying a context identifier (an odd-valued integer),